The Thesis Committee for Andrew Takeshi Takano
certifies that this is the approved version of the following thesis:

Numerical Analysis and Design of Satellite Constellations for Above the Horizon Coverage

APPROVED BY

SUPERVISING COMMITTEE:

Belinda G. Marchand, Supervisor

Wallace T. Fowler

Numerical Analysis and Design of Satellite Constellations for Above the Horizon Coverage

by

Andrew Takeshi Takano, B.S.

THESIS

Presented to the Faculty of the Graduate School of

The University of Texas at Austin

in Partial Fulfillment

of the Requirements

for the Degree of

MASTER OF SCIENCE IN ENGINEERING

THE UNIVERSITY OF TEXAS AT AUSTIN

December 2010

To my parents.

Acknowledgments

First and foremost, I owe my deepest thanks to my supervisor, Dr. Belinda Marchand. Without her guidance, motivation, and financial support I could never have completed this thesis. I greatly appreciate that she brought me into her research group – I cannot imagine a more productive academic environment for myself.

Additionally, I wish to thank Dr. Wallace Fowler, not only for taking the time to review and provide feedback on this work, but also for the wealth of knowledge he has imparted upon me in his space mission design courses. His passion for teaching has been an inspiration to me in my own teaching endeavors.

My gratitude goes to Livy Knox in the Cockrell School of Engineering. Early in my graduate studies, when I found myself without funding, Livy took a chance on me and allowed me to teach undergraduate calculus courses in the General Engineering Department. That experience turned out to be one of the most rewarding I have had in graduate school. Aside from providing funding, I gained a new passion and ability for teaching – things I had failed to attain while grading endless stacks of homework papers and exams.

Of course, I also owe my thanks to the entire Aerospace Engineering Department at the University of Texas for providing the support and a sense of community that is so essential to surviving the stressful trials of graduate school. Many of my professors, including Dr. Maruthi Akella, Dr. Leszek Demkowicz, Dr. Wallace

Fowler, Dr. David Hull, Dr. E. Glenn Lightsey, Dr. Belinda Marchand, and Dr. Cesar Ocampo instructed courses which were essential to my academic development. I feel privileged to have had the opportunity to study under these skilled teachers.

I am grateful to have had officemates, William Hickey, Sara Scarritt, and Divya Thakur who have made my graduate experience more productive and enjoyable, not only by lending their expertise and providing necessary research critiques, but also by being good friends to have a laugh with after a long day. My heartfelt thanks go to my girlfriend, Lisa, for all her love, support, and patience even when I buried myself in my research and worked through long nights and weekends.

Finally, I wish to thank my parents. Their support has been unwavering, and their encouragement unending. I could not have asked for better parents. Without them, I am certain I would have never made it this far.

This research was funded by the Air Force Office of Scientific Research through contract #FA9550-09-1-0227.

Andrew Takeshi Takano

The University of Texas at Austin

December 2010

Numerical Analysis and Design of Satellite Constellations for Above the Horizon Coverage

Andrew Takeshi Takano, M.S.E.
The University of Texas at Austin, 2010

Supervisor: Belinda G. Marchand

As near-Earth space becomes increasingly crowded with spacecraft and debris, the need for improved space situational awareness has become paramount. Contemporary ground-based systems are limited in the detection of very small or dim targets. In contrast, space-based systems, above most atmospheric interference, can achieve significant improvements in dim target detection by observing targets against a clutter-free space background, i.e. targets above the horizon (ATH). In this study, numerical methods for the evaluation of ATH coverage provided by constellations of satellites are developed. Analysis of ATH coverage volume is reduced to a planar analysis of cross-sectional coverage area in the orbit plane. The coverage model performs sequences of boolean operations between polygons representing cross-sections of satellite sensor coverage regions and regions of interest, returning the coverage area at the desired multiplicity. This methodology allows investigation of any coverage multiplicity for planar constellations of any size, and use of arbitrary

sensor profiles and regions of interest. The implementation is applied to several constellation design problems demonstrating the utility of the numerical ATH coverage model in a constellation design process.

Table of Contents

List of Figures

List of Tables

List of Algorithms

Nomenclature

A_{AS}	Total area inside AS
$A_{p\times}$	Area subject to p multiplicity coverage inside AS
AS	Region bounded by the dual-altitude band shell
ATH	Above the horizon
BTH	Below the horizon
$C_{1\times}$	Region of single coverage multiplicity in the region of interest
$C_{2\times}$	Region of double coverage multiplicity in the region of interest
$C_{3\times}$	Region of triple coverage multiplicity in the region of interest
$C_{p\times}$	Region of p coverage multiplicity in the region of interest
GPC	General Polygon Clipping[6] library
h_l	Lower target altitude
h_s	Satellite altitude
h_t	Tangent height
h_u	Upper target altitude
KKT	Karush-Kuhn-Tucker conditions
LTAS	Lower target altitude shell
MEX	MATLAB Executable

m Polygon resolution (integer)

n Number of satellites in a constellation

PPC Points per contour (polygon resolution)

p Coverage multiplicity (integer)

$Q_p(n)$ Number of clip operations for n satellites, analyzed for $p\times$ coverage multiplicity

$q_p(n)$ Number of unique p-tuplet sets for n satellites, analyzed for $p\times$ coverage multiplicity

R_E Mean equatorial Earth radius

RS Satellite sensor region

RS_{arb} Satellite sensor region (arbitrary profile)

RS_E Effective satellite sensor region

$\mathbf{RS}_E$ Array of n effective satellite sensor regions

RS_{TE} Total effective satellite sensor region (combined coverage between one or more satellites)

THC Tangent height cone

THS Tangent height shell

THT Tangent height triangle (in-plane cross-section of THC)

UTAS Upper target altitude shell

α Flight path angle

Γ	Constellation deployment cost
γ	Half the interior angle of the THC/THT
ε	Machine epsilon – distance to the nearest number discernible from 1
ϵ	Tolerance
θ	Angle between r_s and the x axis in the orbit plane

Chapter 1

Introduction

Traditionally, a primary objective in the design of most satellite constellations has been to maximize sensor or communication coverage of targets on the surface of the Earth. Notable satellite constellations focused on providing ground coverage include GPS, TDRSS, Iridium, and Molniya.[8] This type of coverage can be described as 'below the horizon' (BTH) coverage,[9] as the primary region of interest lies below the horizon from the perspective of each satellite.

In contrast, 'above the horizon' (ATH) coverage[10] describes coverage of a region of interest that lies a specified distance above the horizon from the perspective of each satellite. ATH and BTH coverage regions are separated by an imaginary cone that emanates from the satellite and runs tangent to the horizon, enveloping the Earth. This boundary is referred to in the literature as the tangent height cone (THC).[4] The in-plane cross-section of the THC, referred to in this study as the tangent height triangle (THT) is shown in Figure 1.1. In general, the THC and THT both extend to infinity, but the THT is truncated for the purposes of this study at a sufficient distance from the satellite. ATH coverage is of interest if, for example, the satellite utilizes a sensor that is unable to observe targets in the direction of the Earth (due to atmospheric interference, albedo, etc.), or if the sensor provides enhanced dim target detection against a space background.

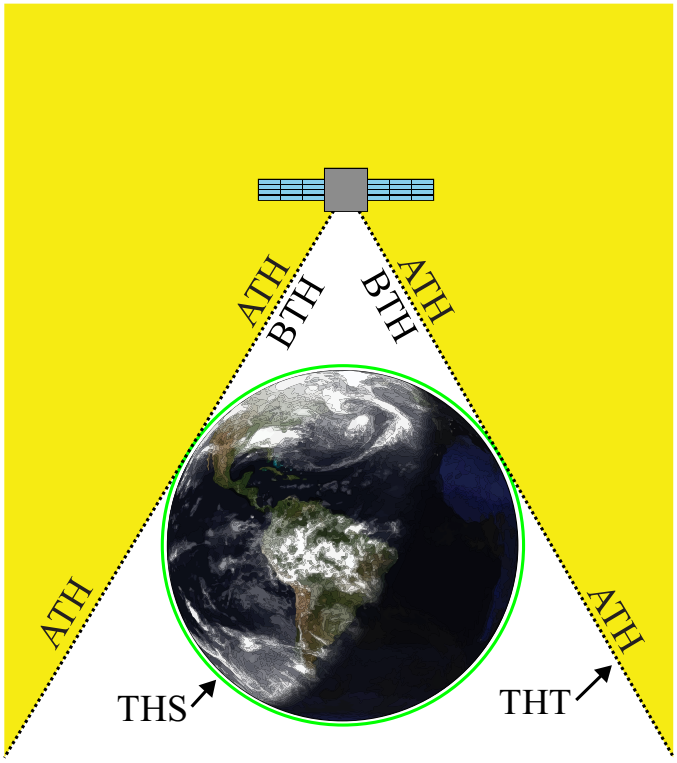


Figure 1.1: ATH vs. BTH Regions – In-Plane Cross-Section

In the current study the 'horizon' is defined relative to a prescribed tangent-height above the Earth's limb that is used to define the Earth-centered tangent height shell (THS), as indicated in Figure 1.1. The tangent height is defined as the closest distance to the Earth's limb where a satellite's sensor is still able to observe objects in space.[4] Thus, any part of the sensor's field of view in the direction of the THS constitutes a blind spot.

Sensors observing targets against a space background rather than a THS background can provide greatly improved detection capability of dim objects.[9] The relative darkness of space allows these difficult-to-detect objects to stand out from their background much more readily. Additionally, constellations designed solely for ATH coverage can benefit from sensors developed especially for detecting objects against a space background.

In this chapter, prior research in the literature regarding ATH coverage is discussed first. Next, the motivation behind the current study is presented, followed by a brief description of the overall approach used. The fundamental and limiting assumptions of the approach are then considered followed by a brief summary of the organization of this thesis.

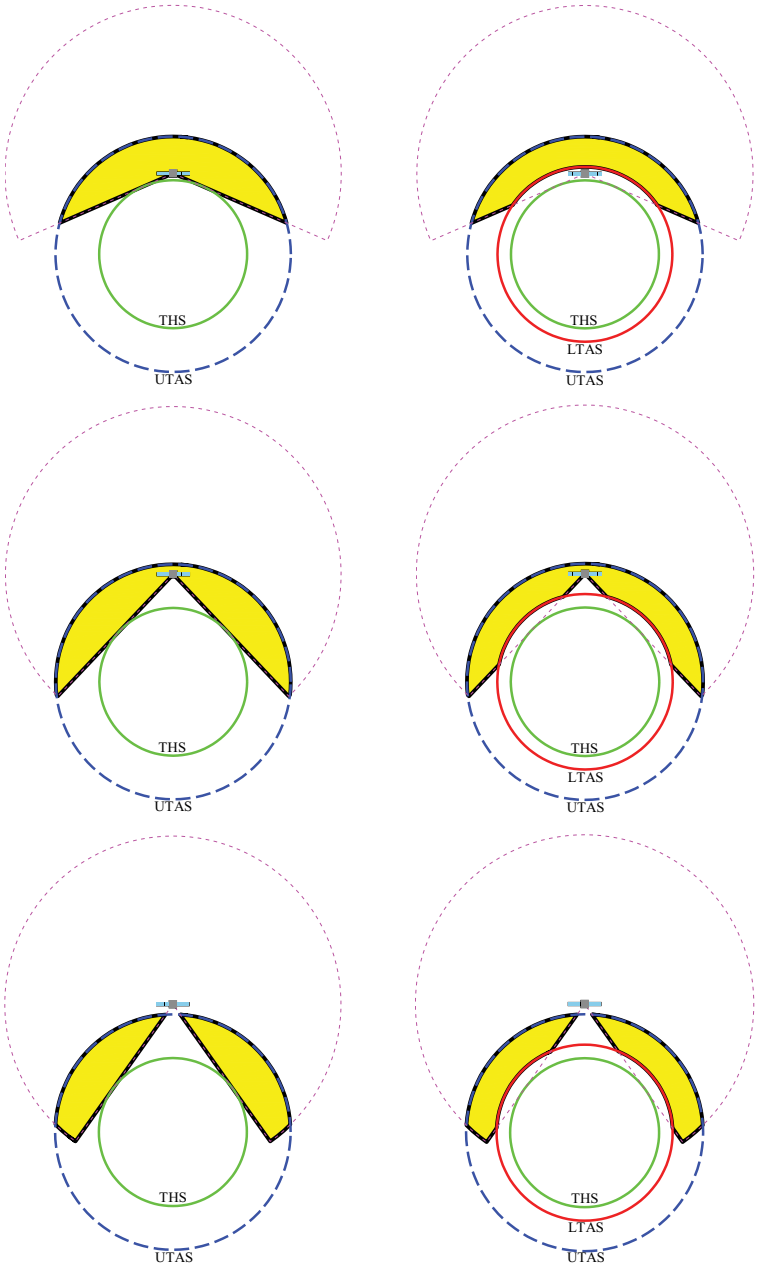
1.1 Background

ATH constellation design methods in the literature can be divided into two categories depending on the characterization of the area of interest – single-altitude and dual-altitude band coverage. Single-altitude band coverage[9–12] analysis only considers the region of interest between the tangent height shell (THS) and the

upper target altitude shell (UTAS). Dual-altitude band coverage[1,4] expands upon single-altitude band coverage by considering an arbitrary lower target altitude shell (LTAS) that is above the THS and below the UTAS. These different scenarios lead to coverage regions of fundamentally different shapes. Several typical single-altitude and dual-altitude band coverage regions provided by a single satellite are shown in Figure 1.2.

1.1.1 Single-Altitude Band Coverage

Multiple investigators have published work concerning ATH coverage[9–12] within the single-altitude band region of interest. An early analysis by Beste[10] considers ATH coverage as a secondary objective to BTH coverage, and does not specifically demarcate between coverage of space above the atmosphere, but still against an Earth or THS background, and coverage against a space background. Rider[9] considers single-altitude band ATH coverage of an equatorially placed constellation and presents case studies and methods of design for constellations providing continuous global ATH coverage of different coverage multiplicities. In the current study, the term 'coverage multiplicity' refers to the number of satellites that are simultaneously covering a particular region. Rider's solutions guarantee global single or double ATH coverage multiplicity, with up to sextuple or octuple coverage near the poles. However, this analysis is performed such that the desired ATH global coverage multiplicity (i.e. single or double) is specified, and the necessary number of satellites and their arrangement for a particular orbital configuration are subsequently determined. As a result, the methods do not consider the actual amount of coverage in



(a) Single-Altitude Band (b) Dual-Altitude Band

Figure 1.2: Examples of Single- and Dual-Altitude Band Coverage Regions

cases where global coverage at the desired multiplicity is not or cannot be achieved. Rider[9] considers the optical design of the necessary sensors in moderate detail but does not describe any methods to specify a range of sensor effectiveness. As a result, constellations designed using these methods benefit from coverage in regions that are very distant from each satellite. In a real case this distance would be constrained, likely due to limited resolving power or sensitivity of available sensors. The mass of a sensor or antenna has been described by Gordon[12] as increasing proportionally with the square of the desired effective range. With this in mind, an assumption of unbounded sensor range is unrealistic. For most applications the optimal constellation configuration will fall somewhere between large constellations of short-ranged satellites at very low altitude and smaller constellations of long-ranged satellites at high altitude. This trade-off is one of the fundamental problems that can be investigated using the numerical methods developed in the current study, by balancing various constellation parameters to more efficiently provide ATH coverage.

Hanson and Linden[11] use a streets of coverage approach to analytically design constellations providing global double BTH coverage and global single ATH coverage. Essentially, every point on the surface of the Earth is visible to at least two satellites and every point in the ATH target region is visible to at least one satellite. Their work is an extension, including consideration of ATH coverage, of previous work done by Rider and Adams,[13] and Walker.[14,15] In their analysis coverage is considered (and, as in Rider's[9] work, ensured by design) based upon projection of coverage onto an imaginary Earth-centered sphere. Due to assumptions of unconstrained sensor range and satellite altitude, as target altitude increases, target

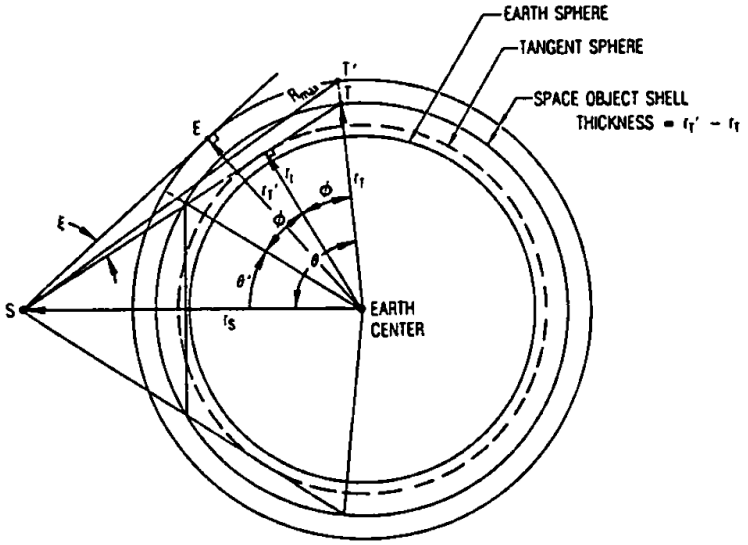
visibility increases as well. Thus, it is reasoned that if the entire surface of the imaginary sphere receives at least single-fold coverage, everything above it also receives single or greater coverage.

Single-altitude band analyses can be considered a subset of the dual-altitude band coverage problem. The methods presented in the current study are demonstrated using a dual-altitude band region of interest, but the same methods can be applied without difficulty to a single-altitude band problem.

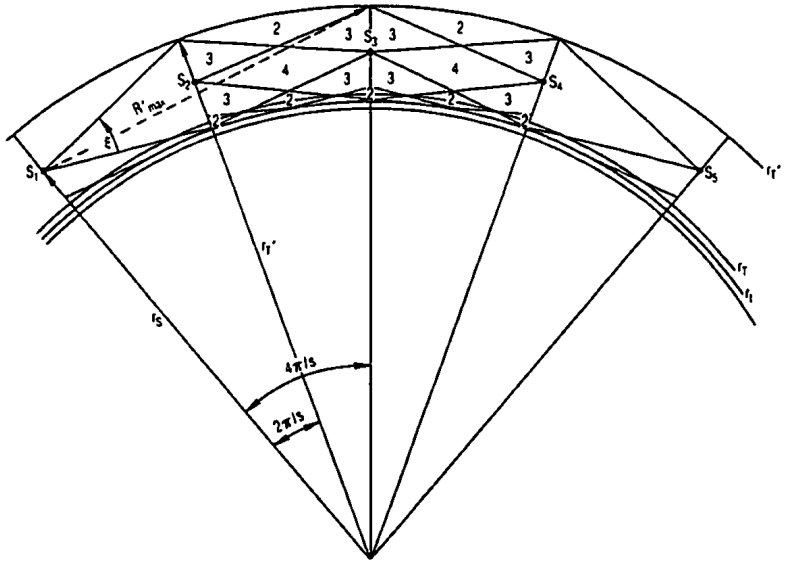
1.1.2 Dual-Altitude Band Coverage

An early treatment of dual-altitude band ATH coverage is presented by Rider[1] who expands upon his earlier work on the single-altitude band ATH coverage problem to include a desired lower altitude bound above the tangent height. The analysis considers various combinations of equatorial and polar orbital planes using geometric reasoning to obtain solutions. Additionally, Rider considers satellite sensors with a constrained field of view, or scan angle, denoted by ξ in Figure 1.3a. An orbit-plane cross-section is shown in Figure 1.3b denoting coverage multiplicities in each region of the target shell where sensor fields overlap between satellites. By placing the satellites in appropriate proximity and ensuring a sufficient field of view and orientation, global ATH coverage can be achieved.

More recent research, published by Marchand and Kobel,[4] presents an analytical coverage model to explicitly evaluate orbit-plane ATH coverage area (the cross-section of the coverage volume) for the dual-altitude band coverage problem. In their research several important simplifying assumptions are made. Satellite



(a) Single Satellite ATH Viewing Geometry



(b) Projection on the Orbital Plane

Figure 1.3: Geometric Design Techniques for Constellations Providing ATH Coverage in a Dual-Altitude Band Region of Interest [1]

sensors are assumed omni-directional, and their analysis focuses on the coverage provided by a single satellite in an Earth-centered circular orbit.

The omni-directional sensor assumption allows the analysis to reduce to a planar problem. By focusing on determining the coverage area in the orbital plane, instead of the coverage volume, the complexities of the three-dimensional problem are avoided. The circular orbit assumption ensures the satellite remains at a constant altitude. When considering only the amount of coverage, rather than at what location above the Earth it occurs, the problem becomes time-invariant.

The analytical coverage model presented by Marchand and Kobel[4] is very useful in that the calculations are computationally low-cost and contain no imprecision beyond the underlying simplifying assumptions (aside from round-off and truncation error in a finite-precision computer implementation). Thus, the resulting coverage model is well suited for use in design problems solved using numerical optimization techniques. In fact, Marchand and Kobel achieve meaningful results in a reasonable amount of runtime using a simple grid search with automated refinement. Their analysis of the coverage provided by a single satellite is an important first step and makes the problem far more tractable. However, coverage in a fully populated constellation inevitably creates regions of overlap with higher multiplicities of coverage than necessary. This means that the coverage provided by a single satellite does not scale intuitively to the total coverage of an entire constellation of similarly positioned and equipped satellites.

The primary difference between Marchand and Kobel's[4] result and earlier ATH coverage analyses[1,9–12] is that an actual measure of coverage is being deter-

mined. Earlier analyses use methods to design constellations that by their very nature ensure the desired coverage characteristics. A model to compute the actual coverage, given a set of parameters, allows for constellation design optimization in a much different way. Perhaps most importantly, it enables design using a variety of generalized numerical optimization methods that already exist.

1.2 Research Motivation

The analytical ATH coverage model developed by Marchand and Kobel[4] provides a very powerful tool for analyzing their problem of interest. Several of the assumptions it was formulated upon are reasonable in many cases. For instance, assumption of omni-directional sensors and circular orbits are a common element in nearly all previous ATH coverage investigations. However, for the design of entire constellations, the capability to analyze coverage for only a single satellite is limiting.

Analytical solutions to complex problems can be very time consuming and labor intensive to produce, assuming one exists at all. Despite their performance and accuracy advantages, analytical solutions are only valid for the specific cases they are formulated to analyze. In contrast, the numerical methods outlined in this thesis enable analysis of a much wider set of problems with fewer constraints. Constellations of many satellites can be analyzed for not only single coverage but any desired coverage multiplicity as well. The burden of relating problem parameters to coverage area can be handled by well-proven numerical methods that allow analysis of interactions between oddly shaped satellite sensor regions and regions of interest. This frees the analysis from the constraint of omni-directional sensor profiles. El-

liptical satellite orbits or constellations with satellites distributed in multiple orbits in the same plane can be analyzed as well, necessitating the introduction of time or a time-like parameter. However, this incurs a significant computational cost and is consequently not explored in the current study. The performance and accuracy penalties associated with a numerical approach to the ATH coverage problem can be quite severe, even for time-invariant analyses. Thorough characterization of these penalties is a recurring objective throughout this thesis.

1.3 General Approach

The approach developed in this study extends the planar analysis developed by Marchand and Kobel.[4] The fundamental problem is to determine combined coverage area within the region of interest, at a coverage multiplicity of interest, between n satellites in a constellation. In this study, the region of interest is the dual-altitude band shell. First, the boundaries of all satellite sensor regions are discretized into polygons. The region of interest is discretized similarly. Next, using several algorithms developed in this study, the satellite sensor regions are processed via sequences of boolean union and intersection operations using the well-developed numerical process of polygon clipping.[16] These operations yield a combined range shell polygon that represents the total region of the desired coverage multiplicity provided by the constellation. This coverage may lie inside or outside (or even inside *and* outside) the region of interest. Finally, a boolean intersection operation is performed between this combined range shell polygon and the region of interest polygon. The area enclosed within this final result polygon is computed numeri-

cally and represents a measure of in-plane ATH coverage at the desired coverage multiplicity within the region of interest.

1.3.1 Fundamental Assumptions

The current study is concerned with developing numerical techniques to analyze satellite constellations for ATH coverage. Most of the available methods for the design of constellations for ATH coverage in the literature[1,9–12] are intended to produce constellations that ensure total coverage by design (with the notable exception of the work by Marchand and Kobel).[4] In contrast, the coverage model developed in this study numerically determines ATH coverage as a function of the constellation parameters in similar fashion to Marchand and Kobel's analytical model.

Keeping in mind that the measure of coverage volume is evaluated by analyzing the cross-sectional coverage area in the orbital plane, the most fundamental assumption is that the constellation in question is planar. This is in contrast to much of the older ATH coverage constellation design techniques that utilize methods similar to the streets of coverage approach commonly used when designing constellations providing ground coverage.[13–15] These works analyze satellites in multiple planes simultaneously, to obtain constellation designs for global ATH coverage. For the current study, because all satellites are limited to a single plane, the relevance of the analysis is limited to ATH coverage within the vicinity of the orbital plane.

Due to the planar nature of the analysis, despite being able to specify more general sensor profiles, coverage must be analyzed under the assumption that there is symmetry in the three-dimensional sensor profiles with respect to the orbital

plane (i.e. the in-plane cross-sections are somehow representative of the three-dimensional case). A good example of a non-omni-directional sensor profile that fits into this context is a conical sensor region extending in-plane from each satellite in the constellation. The in-plane cross-section is a triangle, and as the in-plane triangular coverage region grows, so does the conical coverage volume. A similar example is investigated in Section 4.6. Such a model is interesting in that the satellite look-angles become additional parameters to the coverage model.

The examples explored in the current study are all time-invariant due to the assumption of circular orbits. Time-varying cases are excluded from this study, not because of a fundamental limitation of the methodology that would preclude them, but because of the impracticality of performing such analyses at present. As is shown in later sections, the time-invariant problems alone require a significant amount of computer time to analyze with significant accuracy. To consider a constellation of satellites distributed in a single elliptical orbit, for instance, the coverage model may require evaluation at numerous times throughout an orbit period in order to determine how the ATH coverage evolves. More complex cases, such as constellations of satellites distributed across multiple orbits in the same plane, may not exhibit periodic behavior at all. Such an investigation may require the coverage model to be evaluated throughout a practically unbounded time interval to determine ATH coverage evolution. Thus, time-varying example problems may require several orders of magnitude more computation time to solve than time-invariant cases. By adhering to time-invariant examples, the solutions presented in this thesis are obtained in a reasonable amount of time using the modest computational resources available.

1.4 Thesis Organization

Chapter 2 begins by defining the fundamental polygon regions used throughout this study. Polygon resolution, i.e. the number of vertices on each boundary, is related to approximation error and preliminary guidelines are established that suggest necessary polygon resolution to achieve a desired accuracy. Next, the process of polygon clipping[16] is briefly discussed and several suitable algorithms in the literature[2,17,18] are compared. Finally, the numerical ATH coverage models used to identify regions of particular coverage multiplicities are developed in set notation. Analysis is performed upon these set notation expressions to determine the number of polygon clipping operations necessary for a given model evaluation.

In Chapter 3, the abstract methodology described in Chapter 2 is developed into working implementations. First, several investigated polygon clipping implementations are discussed in detail followed by a performance comparison between them. The fastest clipping implementation, operating natively in C++, is then analyzed in-depth to establish performance guidelines for later studies. The numerical ATH coverage models are implemented and described in pseudo-code. Finally, an investigation is presented into the error and performance observed while solving actual ATH coverage problems. The numerical coverage models are compared to the analytical model developed by Marchand and Kobel.[4]

Chapter 4 begins by developing a simple financial model for use in four parameter optimization-driven example design problems. This financial model is used as either a constraint or an objective function in each example. In the first example, the numerical ATH coverage model is used as an objective function, then

as a constraint function in Example 2. In the third example, the numerical ATH coverage model is used as both a constraint and objective function (each analyzing for regions of different coverage multiplicity). Finally, a constellation of satellites with an arbitrary sensor profile is analyzed to determine a configuration yielding maximum single multiplicity coverage subject to a budget constraint.

Results and conclusions of the study are summarized in Chapter 5. Possible extensions to the full three-dimensional multi-plane case are suggested.

Appendix A presents the rederivation of the analytical coverage model developed by Marchand and Kobel.[4] The resulting implementation of this model is used throughout this research for verification against the numerical models. Appendix B contains example plots that are discussed in passing in Chapter 4 but not presented there for brevity.

Chapter 2

Methodology

This chapter discusses the general approach developed in this study to numerically analyze ATH coverage. As discussed in Chapter 1, the first component of the numerical ATH coverage model is to discretize the sensor cross-sections and region of interest into polygons. These sensor cross-section and region of interest polygons are defined by finite sets of vertices approximating the curvilinear boundaries of the 'true' regions. Following a simple discussion regarding the amount of inaccuracy introduced by approximating curvilinear shapes with polygons, the approaches to defining these sensor cross-section and region of interest polygons are discussed.

Next, polygon clipping, the primary numerical process used to compute ATH coverage in this study, is discussed in moderate detail, and several algorithms are investigated. A brief discussion of reported algorithm performance follows.

Finally, the numerical ATH coverage models developed for this study are presented in detail. For illustrative purposes, the single, double, and triple coverage models are discussed explicitly before generalizing to the arbitrary coverage multiplicity model. Performance considerations for these coverage models are then discussed in detail.

2.1 Region Approximations

Previous work by Marchand and Kobel[4] considers the single satellite case analytically, and as such, the simple regions involved are considered exactly in terms of their geometric parameters. For generalized numerical analyses, however, this approach is impractical as the number of parameters becomes intractable. Thus, a numerical process is sought instead, one that identifies the same information regarding coverage without the need for analytic determination.

2.1.1 Discretization of Non-Linearly Bounded Regions

Almost all interacting regions in this problem (sensor range shells, and altitude shells) feature mostly smooth curvilinear boundaries. As the analysis branches into more general cases featuring non-omni-directional sensors, it becomes practically impossible to express the boundaries analytically. It is, however, convenient to discretize these regions into a finite set of vertices defining their boundaries (i.e. the standard method for defining polygons). Within this context, various robust numerical algorithms can be applied to analyze the interactions (overlap) between different regions using the process broadly referred to as polygon clipping.

2.1.2 Error Analysis

During polygon clipping, contour resolution and polygon corner interactions are the primary factors contributing to the inaccuracy of region approximation. Most of the odd small-scale variations in ATH coverage area encountered in later results are primarily due to corner interactions. The impact upon accuracy of both

contour resolution and corner interactions is mitigated by increasing the contour resolution. This is because a finer resolution yields gradual approximating curves, more accurate to the 'true' curvilinear boundary. Nevertheless, this increase in accuracy is accompanied by a significant performance penalty, as demonstrated later in this study.

2.1.2.1 Contour Resolution

The 'resolution' of a polygon refers to the number of points per contour (PPC) that define its boundary. Polygons may have multiple contours (i.e. a region with an interior hole, or a polygon composed of multiple separate regions), thus, this variable indicates the number of vertices used to define each contour upon generation of the initial polygon. Figure 2.1 shows an example polygon defined at 9 PPC resolution – all contours, whether defining fill or hole regions, are defined by 9 vertices each. How polygon resolution more specifically applies to each of the fundamental polygon shapes is discussed in Section 2.1.3.

In contrast to similar analytical representations, this finite decomposition introduces error inversely proportional to the resolution of the approximating polygons. Because the altitude shells are always spherical in this study and, initially, omni-directional sensor profiles are considered, the non-linearly bounded regions contain contours defined by circular arcs. Thus, analyzing the error introduced in the discretization of a circle provides a good baseline for judging a suitable polygon resolution.

Consider, for example, the approximation of a circle using an inscribed

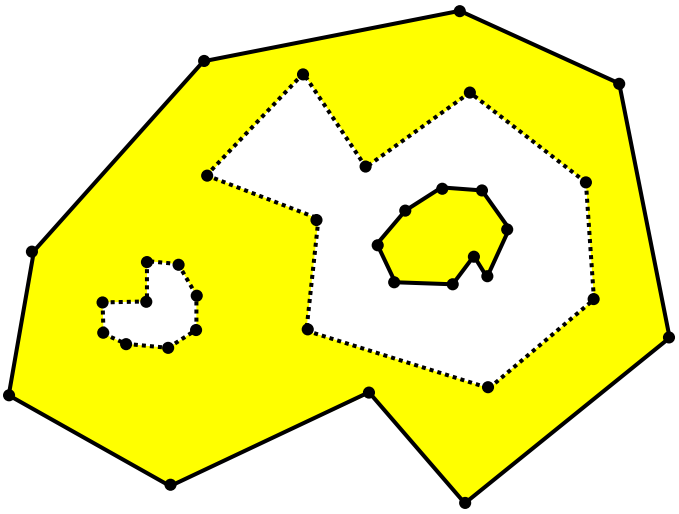


Figure 2.1: All Contours in Each Polygon Defined by m Vertices Each ($m = 9$ PPC Shown)

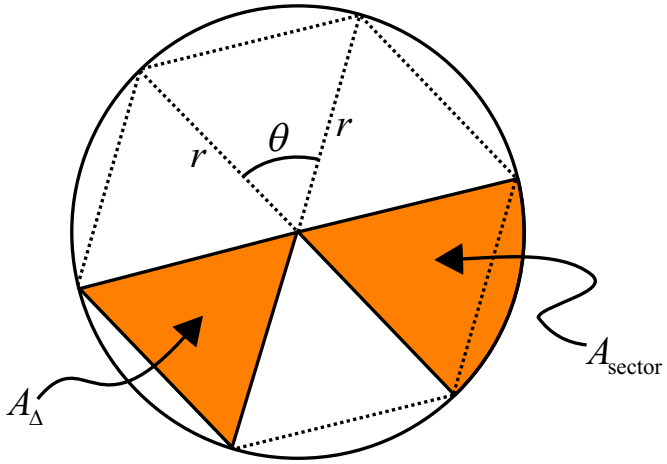


Figure 2.2: Approximation of a Circle With an Inscribed Polygon

hexagon, shown in Figure 2.2. Even more generally, for a polygon with n sides, finding the relative error in approximation of the area computation for any one of the n circular sectors is equivalent to finding the relative error in area computation for the entire circular approximation. Thus, using the notation in Figure 2.2, the relative error is determined as

$$A_{\text{err}} = \frac{A_{\text{sector}} - A_{\Delta}}{A_{\text{sector}}}. \tag{2.1}$$

Given that the interior angle of each sector for an n-sided polygon is

$$\theta = \frac{2\pi}{n}, \tag{2.2}$$

the area of the circular sector is then trivially determined as

$$A_{\text{sector}} = \frac{\pi r^2}{n}. \tag{2.3}$$

Using simple trigonometric arguments,[19] the area of the triangular region (isosceles in general), as a function of the circular radius, r, and the interior angle, θ, is determined by

$$A_{\Delta} = \frac{1}{2} r^2 \sin\theta. \tag{2.4}$$

Combining these two results, and performing some algebraic simplification yields

$$A_{\text{err}} = 1 - \frac{n}{2\pi} \sin\frac{2\pi}{n}, \tag{2.5}$$

where the domain in n is, of course, limited to positive integers greater than three (a polygon with only two linear edges has zero area). Plotting the resulting curve up to $10,000$ PPC, and multiplying by 100% yields the percent relative error in area

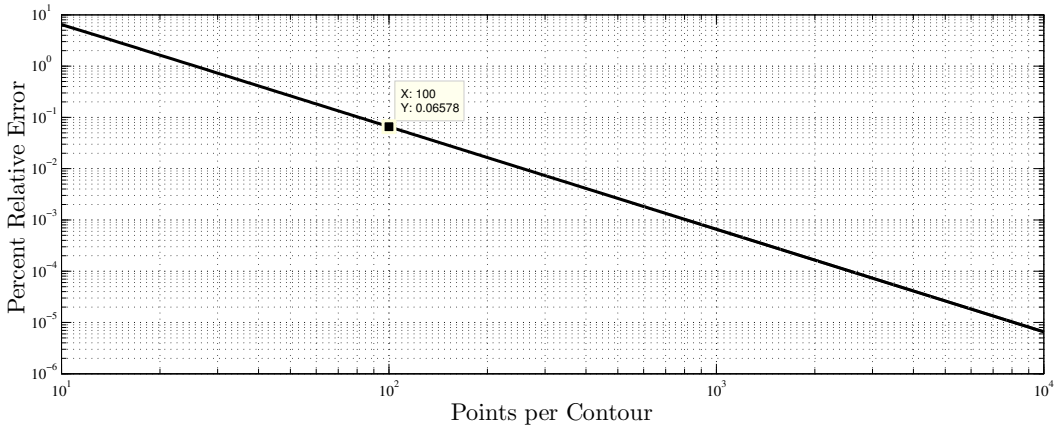


Figure 2.3: Percent Relative Error in Area Calculation of an Inscribed Circular Polygon Representation vs. Contour Resolution

inherent to the polygon representation of an inscribed circle, shown in Figure 2.3. This plot supports the selection of 100 PPC in order to achieve a (admittedly arbitrary) desired accuracy on the order of 0.1% relative error. This result is supported again in later results, shown in Section 3.3.2.

2.1.2.2 Polygon Corner Interactions

When regions are approximated as polygons, curvilinear boundaries are replaced by a series of line segments, with a corner between each. The sharpness of the corner depends upon the curvature of the boundary and the resolution of the approximation. Consider the coverage of three satellites within a dual-altitude band region of interest, shown in Figures 2.4 and 2.5. Cosmetically, the 100 PPC representation, shown in Figure 2.4, appears identical to an analytical representation using true circular arcs. In contrast, the 7 PPC analysis in Figure 2.5 is unacceptably poor. The key observation in the 7 PPC case is that although the three satellite

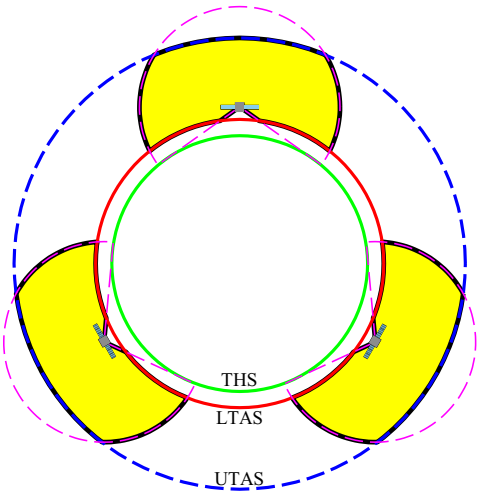


Figure 2.4: Coverage by Three Satellites Computed at 100 PPC

coverage regions (shaded) should be the same by symmetry, they are clearly not equal due to the non-symmetric distribution of corners on the altitude shells across the three coverage areas.

Table 2.1: Area – 7 PPC vs. 100 PPC

Satellite	7 PPC	100 PPC
LL	0.884469	1.03853
Top	0.867410	1.03857
LR	0.833257	1.03853
Total	2.58514	3.11564

In fact, in comparing the computed areas of the shaded polygons, shown in Table 2.1, the 7 PPC case features a large variation in area across the three coverage regions subject to different corner interactions, while the 100 PPC case is far more consistent. Also note that the computed total area in the 7 PPC case differs by 17% from the 100 PPC case, again illustrating the impact of low contour resolution

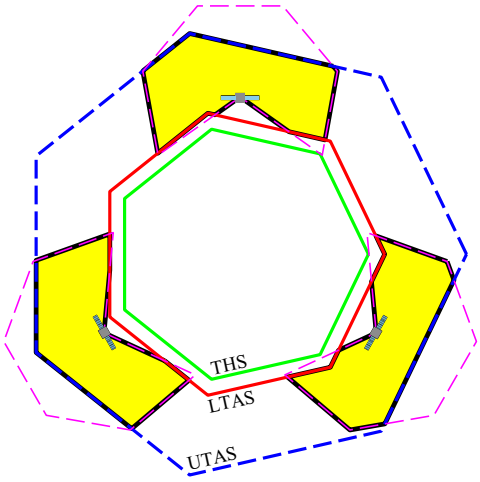


Figure 2.5: Coverage by Three Satellites Computed at 7 PPC

mentioned in Section 2.1.2.1.

2.1.3 Basic Regions

Much of the analysis presented in this thesis can be considered in terms of two fundamental region types – the effective range shell, and the dual-altitude band shell. For simplicity, initial analyses performed in this investigation assume omni-directional sensor profiles with no loss of generality.

2.1.3.1 Effective Range Shell, RS_E – Omni-Directional Sensor

By assuming an omni-directional sensor profile, the vertices defining the in-plane cross-section, the effective range shell (RS_E, the *actual* ATH region covered by the satellite, i.e. the range shell excluding the tangent height triangle (THT) region) are easily computed based solely on the satellite parameters. This simplification is

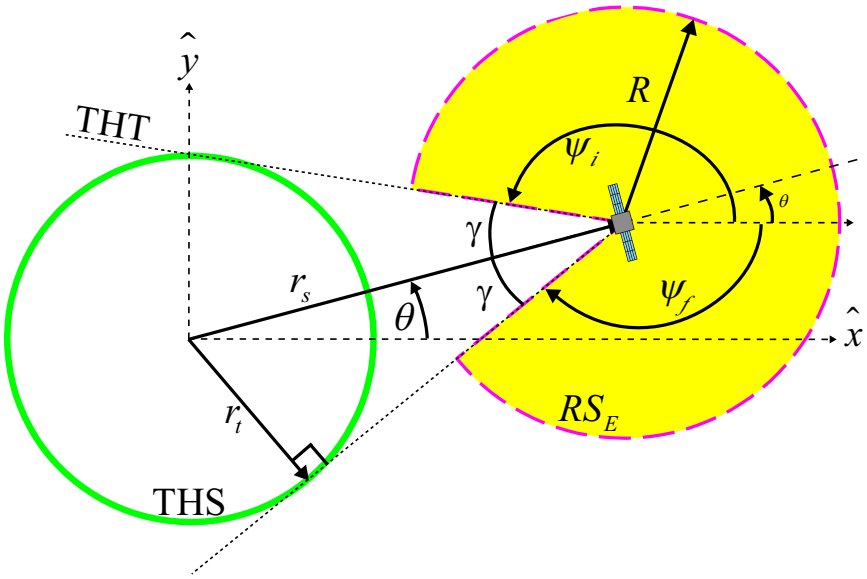


Figure 2.6: Notation for Computing Effective Range Shell (RS_E) Vertices for the Omni-Directional Sensor Case

possible due to the circular boundary of the omni-directional sensor cross-section. Effective range shells are shown (but not labeled) in Figures 2.4 and 2.5 as dashed boundaries centered on each satellite. A notated illustration of the effective range shell is shown in Figure 2.6.

The interior half angle of the THT, γ, is computed as

$$\gamma = \arcsin \frac{r_t}{r_s}. \tag{2.6}$$

Given an in-plane longitude (angular displacement of the satellite from the positive x-axis) of θ, it is then clear that the initial and final angles for the circular portion

of the boundary are given by

$$\psi_i = \pi - \gamma + \theta, \tag{2.7}$$

$$\psi_f = -\pi + \gamma + \theta. \tag{2.8}$$

Consider an approximation of the range shell in terms of an inscribed polygon with a resolution of m PPC. The determination of these m vertices requires that the data structure be populated first with a single point at the location of the satellite at $(x, y) = (r_s \cos\theta, r_s \sin\theta)$, followed by $m - 1$ equally distributed points along the circular portion of the boundary in the clockwise direction. The curve is implicitly closed between the final and initial vertices.

To define all the intermediate points along the circular portion of the boundary, between ψ_i and ψ_f, it is useful to define an intermediate value, ψ_{int}, as

$$\psi_{\text{int}}(j) = \psi_i + \frac{(j-2)(\psi_f - \psi_i)}{m-2}, \tag{2.9}$$

where j is an integer monotonically increasing from 2 to m in increments of 1 (with $j = 1$ referring to the first vertex, defined as $(r_s \cos\theta, r_s \sin\theta)$). Thus, a programming loop is used to iterate and produce coordinates for the effective range shell using the equations

$$(x_1, y_1) = (r_s \cos\theta, r_s \sin\theta) \qquad j = 1, \tag{2.10}$$

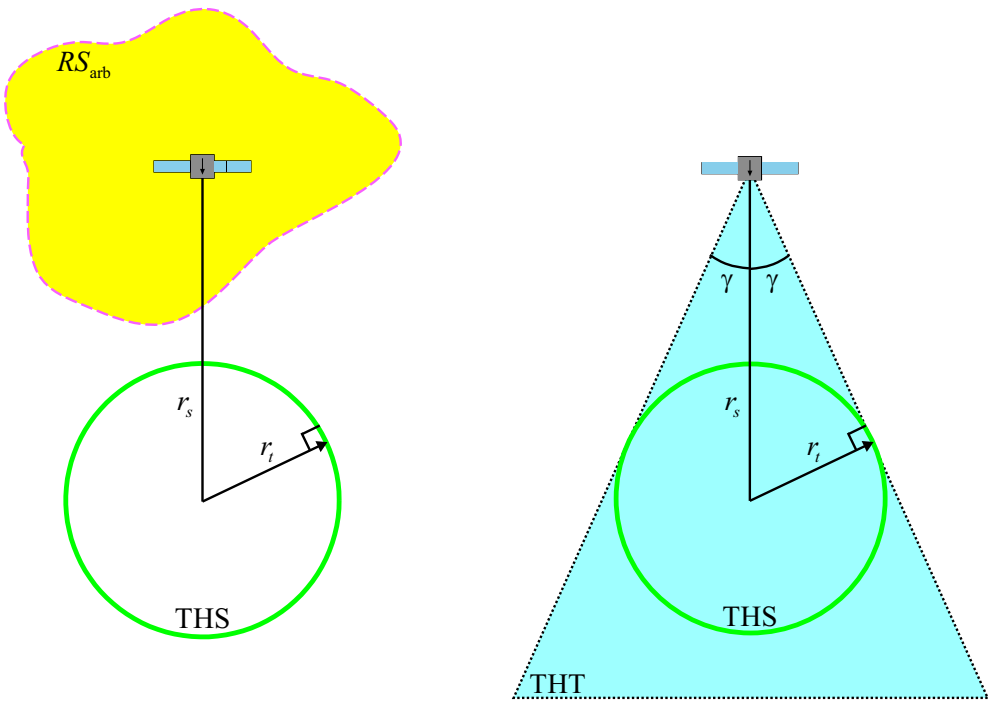
$$(x_j, y_j) = (R\cos\psi_{\text{int}}(j) + x_1, R\sin\psi_{\text{int}}(j) + y_1) \qquad j = 2, 3, ..., m. \tag{2.11}$$

2.1.3.2 Effective Range Shell, RS_E – Arbitrary Sensor Profile

When analyzing non-omni-directional sensor profiles with a known in-plane cross-section, the effective range shell, RS_E can still be isolated, but at a cost of

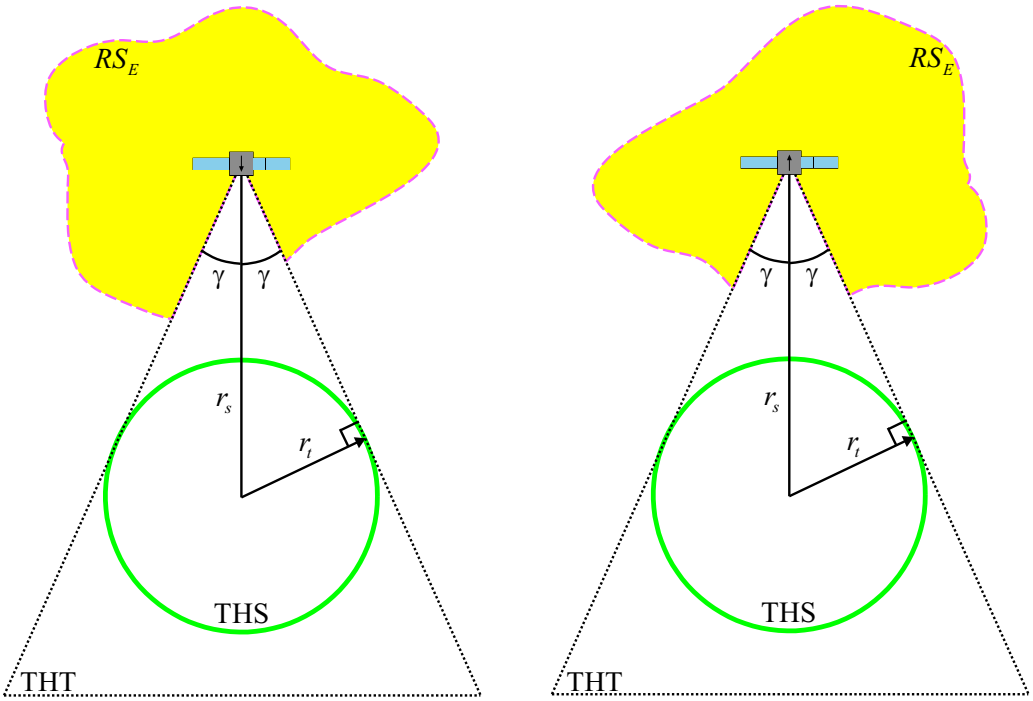
performing a boolean operation on the involved polygons (using polygon clipping, discussed in 2.2). Referring to Figure 2.7, a polygon is defined representing the in-plane cross-section of the arbitrary sensor profile, RS_{arb} and the tangent height triangle, THT. The THT need only extend to a distance from the satellite greater than the maximum extent of the sensor cross-section. This requirement is imposed so that after the clip operation, there are no remaining regions of the sensor cross-section in the direction bounded by the sides of the THT. To obtain the effective sensor range shell, RS_E, the THT is differenced (using a polygon clipping operation) from RS_{arb}. Because of the non-omni-directional shape, the orientation/attitude of the sensor cross-section becomes a parameter, and the resulting effective range shell shape can vary considerably depending on this new parameter, as shown in Figure 2.8.

Performing this additional clipping operation incurs a performance penalty over the omni-directional case, where none is required. However, the runtime of such a clipping operation will be far less than the typical clipping operations performed in this study. The THT polygon consists of only three vertices, and performance of the most efficient clipping algorithm considered (developed by Martínez)[2] is $O((l + k) \log l)$ for l vertices total, and k intersections. Thus, for a reasonable polygon resolution, necessitating the addition of three vertices and only two intersection points (typically), runtime is considerably less than for a clipping operation between two polygons of like-resolution.



(a) RS_{arb} Shaded (b) THT Shaded

Figure 2.7: RS_{arb} and THT: THT is Subtracted From RS_{arb} to Form RS_E



(a) 0° From Nadir Direction

(b) 180° From Nadir Direction

Figure 2.8: RS_E for RS_{arb} at Different Satellite Attitudes

2.1.3.3 Dual-Altitude Band Shell

The annulus referred to as the dual-altitude band shell, AS, shown in Figure 2.9, is centered at the origin (center of the Earth), and has an inner radius of the lower target radius r_l, and an outer radius of the upper target radius r_u. In the implementation used for this thesis, an altitude shell created for some resolution given by m PPC, contains m vertices on the inner boundary, and m vertices on the outer boundary.

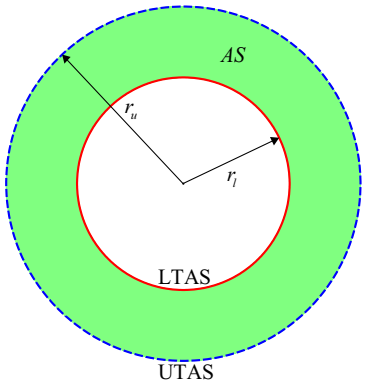


Figure 2.9: Dual-Altitude Band Shell Polygon Region, AS

2.2 Polygon Clipping

As discussed in the preceding material, the planar-case problem of analyzing ATH coverage of satellites with arbitrary sensor profiles (with known in-plane cross-sections) can be reduced to the numerical problem of boolean operations on polygons, that can solved using the process of polygon clipping. Thus, in the course

of implementing these techniques, it is worthwhile to understand not only how they work, but also which polygon clipping algorithms are the most robust and produce the best performance for the problem of interest. Many constellation design problems are well suited to a parameter optimization approach, that, by nature, requires a large number of model evaluations. As will be shown later, the vast majority of numerical ATH coverage computation time is consumed by the polygon clipping process, thus, performance is a primary concern.

2.2.1 Background of Polygon Clipping

Polygon clipping is essentially the act of 'cutting' one polygon (the subject polygon) with another (the clip polygon). By analyzing the interaction of the two shapes, various Boolean operations are carried out. In the context of the ATH coverage problem, unions, intersections, and differences are of primary interest.

The origins and continued usage of polygon clipping primarily reside in computer graphics,[18] electrical circuit design,[16] and the study of geosciences.[2] In computer graphics, the primary motivations are to identify and omit off-screen polygons from the rendering process (thus improving performance), as well as removing segments of a polygon that should be obscured by other objects (i.e. hidden-surface rendering of scenes). Circuit designers sometimes use polygon clipping to numerically verify complex designs, ensuring that no electrical traces are unintentionally bridged during the design and revision process. In the study of geosciences, geospatial data (i.e. annual polar ice cap formations), is often stored in the form of polygons. Researchers perform various Boolean operations upon these data sets to make

comparisons or analyze variations over time.

Applications to computer graphics shaped the first polygon clipping algorithms.[16] For instance, early algorithms only considered rectangular clip polygons (i.e. representing the boundary of a computer screen). Over time, algorithms were developed to handle convex and, eventually, non-convex clip polygons. A convex polygon is one where any two interior points can be connected by a straight line without crossing the polygon's boundary. An example of a convex polygon is a rectangle, circle, or ellipse, as opposed to a non-convex polygon such as an annulus, or crescent shape.

A generalized system was first developed by Weiler (1980),[20] capable of handling non-convex clip and subject polygons, and polygons with self-intersections. This algorithm is not commonly used today because of its relative poor computational performance, and reliance upon a very complex data structure that makes implementation tedious.[18] The next significant improvement was made by Vatti (1992).[17] His algorithm is much more intuitive than Weiler's, and offers a significant performance improvement. Since then, algorithms by Greiner and Hormann (1998),[18] as well as Martínez and Rueda, et al (2009)[2] were published, both exhibiting new ways to handle Boolean operations on arbitrary polygons.

Algorithms by Vatti, Greiner, and Martínez are all major milestones, and are thus the only three discussed in this study. Vatti's algorithm has attained widespread usage, and is the basis for several of the freely available clipping libraries in existence.[6,21,22] Greiner and Hormann's algorithm, while providing improvements in performance, lacks robustness for reasons discussed in Section 2.2.3.2, and has

thus not yet gained widespread implementation. The result published by Martínez is elegant and robust, but it is still new and has not yet gained widespread implementation. In addition to its robustness, Martínez' method provides a substantial (order of magnitude) improvement in performance over Vatti's algorithm according to its creator.

2.2.2 Location of Intersections

Any introductory course in computational geometry initially addresses the problem of counting and locating the intersections among a finite set of randomly oriented and placed line segments in a plane. This topic proves to be quite instructive for the student, as the efficient algorithm that is ultimately explored is the so-called 'plane-sweep' algorithm;[16] a type students of computational geometry frequently encounter, especially in polygon clipping methods. Essentially, a plane-sweep algorithm considers a geometric data set by 'sweeping' through the entire set in one geometric direction performing various comparisons along the way.

All polygon clipping methods also require knowledge of polygon edge intersections – in general, they form the necessary corners of result polygons where the subject and clip polygon boundaries intersect. The clipping algorithms presented by Vatti[17] and Martínez[2] both specify methods to locate intersection points concurrently with the clipping process, in a method almost identical to that discussed here. Greiner's method[18] leaves this important component unspecified and some other external method must be supplied during implementation. The primary reason for this omission is that Greiner's method is fundamentally different from the

other two that are plane-sweep algorithms themselves. Determination of edge intersections must be addressed in order to achieve accurate clipping solutions, as is demonstrated at the end of this section.

Given a set of two polygons that are to be clipped, defined by a combined total of l vertices (and consequently, l edges), a simple check-all algorithm may be implemented that checks every edge against every other edge (including the possibility for self-intersection of a polygon boundary). The total number of unique combinations to check, skipping comparisons of edges against themselves, is given by

$$\frac{l^2 - l}{2}. \tag{2.12}$$

However, since (to achieve desired accuracy, as discussed earlier), l is typically large, the resulting performance of this approach is $O(l^2)$.[16] This is extremely inefficient, especially in cases where intersections may be relatively sparse compared to the number of polygon edges (as is the case for the problems analyzed in this thesis).

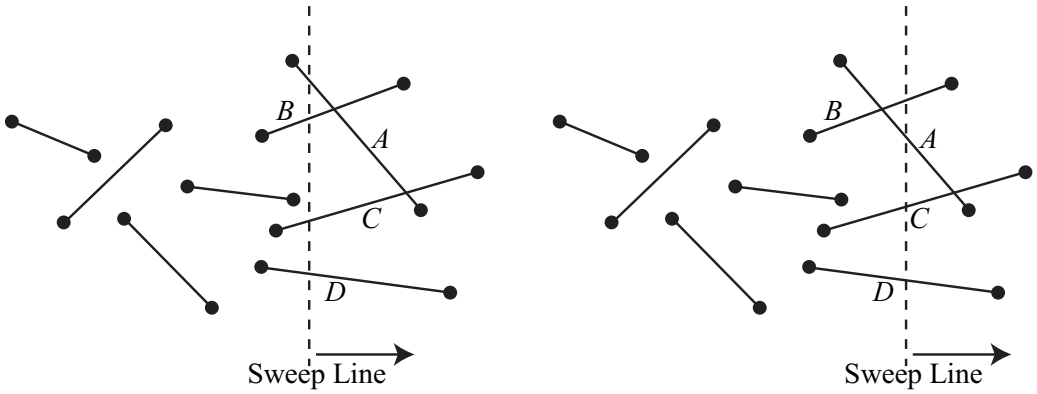
Fortunately, more efficient algorithms have been developed. One that has achieved widespread usage was put forth by Bentley and Ottmann (1979),[23] handling the general problem of locating intersections for l line segments in a plane. The author demonstrates that for l line segments with k intersections, a lower bound in performance of $O(l \log l + k \log l)$ is achievable through this algorithm. In the context of the present research, the number of intersections is quite small, typically less than 8 for problems involving simple geometries. For typical l values of 100 or more, this algorithm offers a substantial improvement over the simple check-all case.

The basic premise of this algorithm is to more intelligently perform intersection checks based on what is geometrically feasible. The endpoints of each line segment are inserted into an ordered data structure (such as a binary search tree),[16] ordered in one primary geometric direction, i.e. increasing in x. Each endpoint is an 'event,' and the 'sweep line' moves from one event to the next taking advantage of the sorted order to make one 'sweep' across the coordinate plane.

Degenerate cases, such as when endpoints or line segments overlap must be specifically accounted for during implementation; this is the primary difficulty in producing a robust implementation, and is by no means a trivial task. Thus, it is beyond the scope of the current conceptual discussion.[23]

Associated with the sweep line is another ordered data structure (linked list or binary search tree) that keeps track of the 'active' line segments, i.e. the line segments that intersect the sweep line at a particular instant. When the sweep line encounters a left endpoint, a segment pointer is added to the sweep line structure. These pointers are sorted in order of their segment intersection with the sweep line, i.e. bottom to top. When the sweep line encounters a right endpoint of one of these segments, the segment pointer is deleted from the sweep line data structure.

The ordered nature of this sweep line data structure is what enables excellent savings in computation – only segments that are 'neighbors' (i.e. physically adjacent) are capable of intersecting (except in certain degenerate cases).[23] Thus, the algorithm only checks these neighbors against one another for intersection, saving a large number of calculations in all but the worst-case scenario (every line segment intersecting every other line segment). Two snapshots of this process are shown in



(a) Sweep Line at x_1 (b) Sweep Line at x_2

Figure 2.10: Sweep Line at Mid-Sweep

Figure 2.10.

At the instant shown in Figure 2.10a, the sweep line data structure contains the ordered elements $\{D, C, B, A\}$. Thus, segment C is checked for intersections with its neighbors, D and B, but not A, or any of the unnamed segments that are now behind the sweep line. Similarly, segment B is checked for intersections with neighbors C and A, (this check finds the intersection with A).

After the sweep line passes the intersection between B and A (shown in Figure 2.10b), the sweep line data structure contains $\{D, C, A, B\}$. Now, A is checked against C (because they are neighbors on the sweep line) and the algorithm locates the intersection between them. Considering that each line segment is only checked against several other line segments, it is clear how this algorithm is a great improvement over the simple check-all algorithm.

While there are newer and slightly faster algorithms than the Bentley and

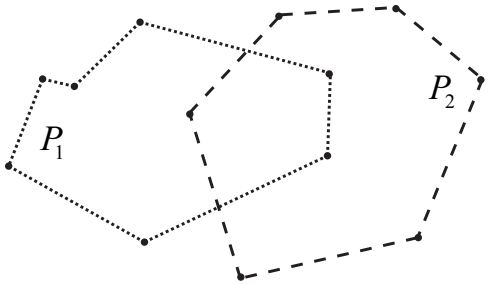


Figure 2.11: Example Overlapping Polygons

Ottmann method,[23] it is still favored for its simplicity, straightforward implementation, and efficient memory use.[16] It is critical that a clipping implementation utilizes an efficient subroutine for locating edge intersections – a study by Greiner[18] shows that even with efficient algorithms, up to 80% of CPU time during polygon clipping operations is spent checking for and locating polygon edge intersections.

All polygon clipping algorithms must, in some way, locate the intersections between line segments in order to produce a geometrically valid result. It is possible to implement a poorly conceived clipping method that ignores these intersection points, and simply determines if each vertex of each input polygon is an interior or exterior point to the other polygon (i.e. using the concept of a winding number as discussed by Greiner).[18] However, the resulting region will generally be incorrect. During a union, for example, the resulting boundary would be incorrectly determined to only span the vertices that are outside the opposing polygon. This approach ignores the boundary contribution of the intersection points. For example, consider the two polygons P_1 and P_2 shown in Figure 2.11. If the union of P_1 and P_2 is performed with this poorly conceived clipping implementation, the resulting

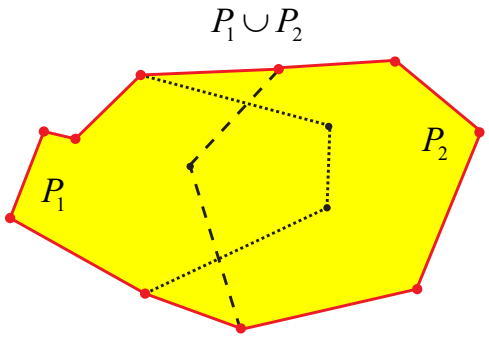


Figure 2.12: Union, Without Edge Intersection Detection

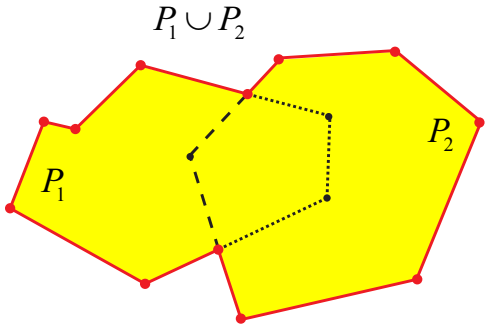


Figure 2.13: Union, With Edge Intersection Detection

polygon will not follow the boundary as intended. This case is shown in Figure 2.12. In actuality, the desired union must incorporate the intersection points, as shown in Figure 2.13.

2.2.3 Clipping Algorithms

Although there are numerous clipping algorithms in the literature, the three discussed here are among the few that were conceived to handle arbitrary subject and clip polygons. The earliest published example, by Weiler,[20] is omitted from this

discussion due to its relative poor performance and difficult implementation.

2.2.3.1 Vatti's Method (1992)

Vatti's method[17] proceeds along the same lines as the previously discussed Bentley and Ottmann[23] line intersection detection algorithm (although Vatti made the arbitrary choice of sweeping top to bottom instead of left to right). When a segment's terminal endpoint is encountered by the sweep line, Vatti's clipping method performs some additional tasks that differentiate it from the simple edge intersection detection algorithm by Bentley and Ottmann. Because the clipping process is not only interested in edge intersections, but also in producing a result polygon, as line segments are removed from the sweep line, they are considered for inclusion in the result polygon. It is in this way that the result polygon is sequentially constructed during the sweep.

The algorithm is considered to be robust, once all degeneracies have been dealt with. However, the original literature is vague on how to handle some, such as horizontal line segments (parallel to the sweep line).

Given the length of time since introduction, speed, and easy implementation (relative to Weiler's algorithm),[20] there have been a number of libraries developed using this algorithm.[21,22] Perhaps most notable is, Murta's General Polygon Clipping (GPC) library[6] which is widely used, and is the clipping library used to produce most of the results presented in this thesis. The GPC library, while thoroughly proven and robust, has the added advantage of being freely available for non-commercial use.

2.2.3.2 Greiner's Method (1998)

In contrast to the plane-sweep method, Greiner and Hormann[18] allow an approach that is quite different in concept, but is arguably more intuitive, and with a data structure that is simpler to implement. Greiner and Hormann provide an analogous physical scenario: starting with the subject polygon, imagine a paint-distributing cart being pushed along the boundary from one vertex to the next. If the initial vertex lies inside the clip polygon, the flow of paint is turned on. The flow of paint is toggled off or on every time the cart crosses the boundary of the clip polygon. A similar process may be repeated, following the boundary of the clip polygon, and painting the portion of the boundary that lies within the subject polygon. The resulting painted and unpainted boundaries are then combined to find the intersection of the polygons (the painted boundaries), or the union of the polygons (the unpainted boundaries).

Translating the paint-cart example into a method that can be implemented, Greiner and Hormann define a simple data structure. Due to the sequential nature of the analysis in this algorithm, the data structure is efficiently implemented as a doubly linked list,[18] rather than the more complex balanced binary search trees[16] necessary for other methods.[2,17] Each entry in the linked list contains the coordinates of the vertex, and pointers to the next and previous vertices on the boundary. The entries also contain an entry/exit switch that instructs the imaginary cart-pusher to turn the flow of paint on or off as needed. This last item is determined by geometric criteria (the winding number, as defined by Greiner and Hormann)[18] and can be stored at the time of edge intersection detection using a method such

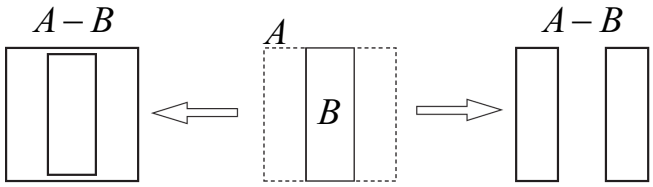


Figure 2.14: Difference Between Polygons A and B With Two Different Perturbation Directions on the Collinear Segments[2]

as that described in Section 2.2.2. Because the algorithm is not of the plane-sweep type, the clipping process and edge intersection determination cannot be performed concurrently. Thus, edge intersections must be located prior to commencement of the clipping process.

One noticeable drawback of the Greiner and Hormann algorithm[18] is the method used to handle the degenerate case of a vertex coincidentally bisecting another line segment. Greiner and Hormann suggest that the vertex location should be perturbed slightly in some arbitrary direction so that it is no longer on top of the line segment. This can cause unpredictable results, as illustrated by Martínez[2] in Figure 2.14. Different choices of vertex perturbation direction can lead to drastically different result polygons. Greiner's method,[18] despite its conceptual simplicity, has yet to gain widespread implementation. Another result published by Liu, et al,[24] proposes several optimizations to Greiner's original method, but does not address the questionable vertex perturbation approach to the aforementioned degeneracy.

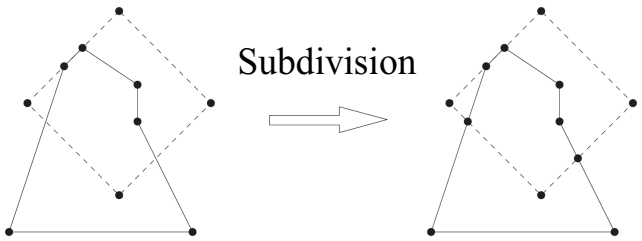


Figure 2.15: Subdivision of Edges of Polygons at Their Intersection Points[2]

2.2.3.3 Martínez' Method (2009)

Similar to Vatti's method,[17] Martínez' method[2] utilizes a plane-sweep approach. One primary difference is the method used to handle edge intersections, reducing issues of degeneracy. Each time an edge intersection is encountered during the sweep, the algorithm subdivides each segment creating new segments to replace the old. This process is illustrated in Figures 2.15 and 2.16. Because line segments then only intersect at endpoints, this subdivision process allows degeneracies to be handled in a much more elegant and exact fashion than Greiner and Hormann's method[18] of vertex perturbation.

The data structure used to organize the edges intersecting the sweep line is virtually identical to that used by Vatti,[17] however it differs slightly in the specification of edge intersection testing criteria. This difference, Martínez claims, is the primary factor behind the algorithm's improved performance.

2.2.3.4 Clipping Algorithm Performance Comparison

For the simple reason that speed is such a crucial factor in almost all applications of polygon clipping, virtually every published article describing a new method

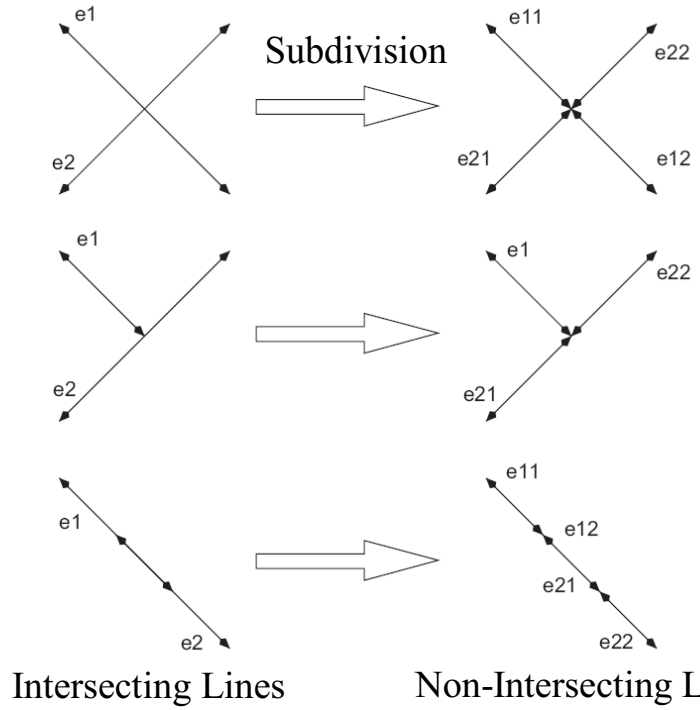


Figure 2.16: Types of Intersections That Lead to a Subdivision[2]

includes a section dedicated to computational performance. Unfortunately, these studies often feature incomplete or sub-optimal implementations of competing algorithms. For instance, both Liu[24] and Martínez[2] compared their algorithms against Vatti[17] and Greiner.[18] However, it appears both studies used a simple check-all algorithm for determining edge intersections when implementing Greiner's algorithm, while comparing it against Murta's GPC library[6] as an example of Vatti's algorithm. As discussed before, Vatti's algorithm uses a plane-sweep method, and thus can concurrently use Bentley and Ottmann's efficient algorithm[23] to locate edge intersections.

As mentioned in Section 2.2.2, Greiner and Hormann's work[18] specifies no method for intersection determination, and leaves the reader to their own devices. The possibility of using a simple check-all method is mentioned in passing, but certainly this approach leaves the Greiner algorithm at a severe disadvantage against Vatti,[17] Liu,[24] and Martínez[2] – all of which specify methods identical to the Bentley-Ottmann[23] algorithm discussed earlier. It should come as no surprise that the Greiner algorithm fares poorly in the studies reported by Liu and Martínez.

With the aforementioned caveat in mind; the published results may provide a valid comparison of their respective algorithms against Vatti's method,[17] but they do not produce a clear picture of how Greiner's algorithm[18] performs. Despite this, comparisons of performance of the other algorithms can still be made. For polygons with large numbers of vertices, Vatti's algorithm demonstrates its superiority relative to Liu's newer algorithm,[24] but not Martínez' algorithm,[2] which is an order of magnitude faster for very large problems. However, Martínez' algorithm performs

only marginally faster than Vatti's algorithm for smaller problems, such as those considered in this research.

Given the high performance afforded by Martínez' algorithm,[2] combined with its subdivision method to handle certain degenerate cases, it seems to be the obvious choice for use in this study. However, at the time of this writing, no thoroughly vetted libraries based on Martínez' method are publicly available. Early in the research process discussed in this thesis, a substantial amount of time was dedicated to developing an implementation of Martínez' algorithm, but it has yet to become fully robust. A number of degenerate cases must still be addressed, and the implementation must then undergo extensive testing before it can be applied to the current research problem. A single oversight or shortcoming in the implementation could cause a long-running parameter optimization analysis to crash, causing a loss of intermediate data, or worse, return erroneous data. Despite this, the time spent was not fruitless – implementation of a polygon clipping algorithm itself is a very useful exercise in developing intuition for problems involving polygon clipping. Lacking a higher performance option with the necessary robustness, Murta's GPC library[6] is utilized for the investigations presented in this thesis.

2.3 Numerical ATH Coverage Models

This section presents numerical ATH coverage models for evaluating amounts of coverage at various desired coverage multiplicities. Within the framework of boolean operations, evaluated using polygon clipping, the resulting expressions are quite intuitive. A problem of particular interest is the ATH coverage provided by

a constellation populated by n satellites with omni-directional sensors, equally distributed within a single circular orbit. Analysis of this problem is straightforward to implement (in large part because it is time-invariant), and is considered for illustrative purposes in this section. It is fundamental to observe that the coverage models presented here can actually apply to *any* configuration of satellites subject to the planar analysis this study is concerned with. Analysis need not be restricted to omni-directional sensors, or even any uniform or regular relationships between multiple orbits satellites could be distributed in, only that the orbits lie in the same plane and satisfy the necessary sensor profile symmetry across the orbital plane to allow reduction to a planar analysis.

2.3.1 Single Coverage

A graphical illustration for the derivation of the single coverage model is shown in Figure 2.17. To begin computing the region of single coverage, $n-1$ union operations (Figures 2.17a-2.17i) are performed between the n satellite effective range shells, RS_E to produce a total effective sensor region, RS_{TE}. A final intersection operation is performed between this total effective sensor region, and the altitude shell, AS (region of interest), yielding the region of total single coverage within the region of interest, shown in Figure 2.17l as $C_{1\times}$. This process is written in set notation as

$$C_{1\times} = \left(\bigcup_{i=1}^{n} RS_{E_i} \right) \cap AS. \tag{2.13}$$

The set-notation expression for $C_{1\times}$ only provides the resulting region of single ATH coverage that, in this implementation, is a polygon defined by a finite

set of vertices on its boundaries (the polygon may be composed of multiple disparate regions). Determination of the enclosed area is then computed from the vertices of the polygon. The method used in this study proceeds as follows – given a set of m vertices along a closed contour expressed as $(x_1, y_1), (x_2, y_2), ..., (x_m, y_m)$, the area enclosed within that contour is determined by the expression[25]

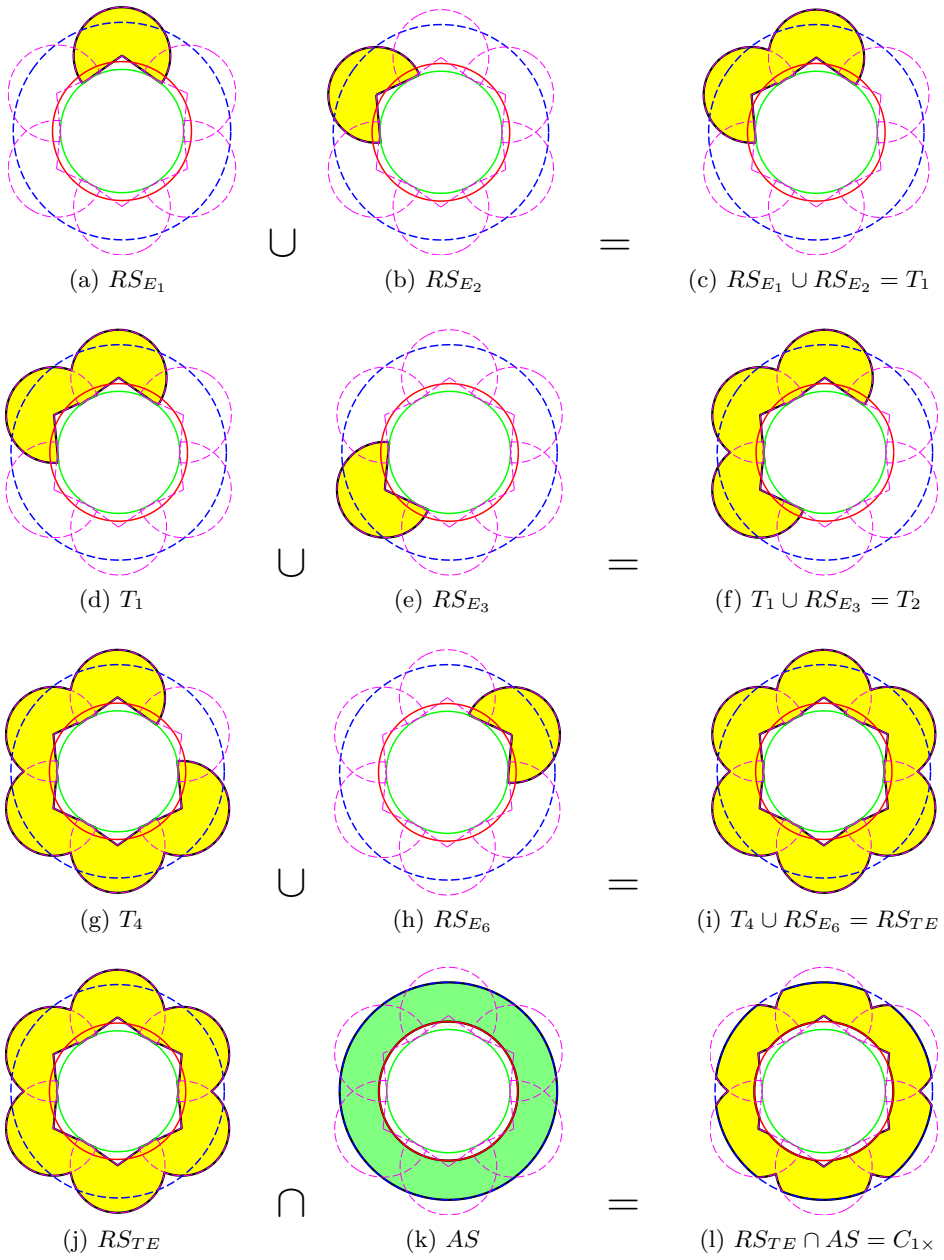
$$A = \frac{1}{2}\left(\begin{vmatrix} x_1 & x_2 \\ y_1 & y_2 \end{vmatrix} + \begin{vmatrix} x_2 & x_3 \\ y_2 & y_3 \end{vmatrix} + ... + \begin{vmatrix} x_m & x_1 \\ y_m & y_1 \end{vmatrix} \right). \tag{2.14}$$

This method is applied for each contour of the resulting polygon $C_{1\times}$, adding areas inside contours denoting fill regions, and subtracting areas inside contours describing hole regions.

2.3.2 Double Coverage

Regions of double coverage are determined in similar fashion, but using a different sequence of Boolean operations. Figure 2.18 shows the process for determining the region of double coverage. First, individual regions of double overlap between sensor regions are identified by performing intersection operations between all unique pairs of sensor regions, as shown in Figures 2.18a-2.18f. A sequence of union operations then combines all individual regions of double coverage, as shown in Figures 2.18g-2.18j. Finally, just as in the single coverage case, an intersection is performed between the total effective sensor region and the altitude shell, to yield $C_{2\times}$ (the region of double coverage within the region of interest), shown in 2.19l. This process is expressed in set notation as

$$C_{2\times} = \left(\bigcup_{i_1=1}^{n-1} \bigcup_{i_2=i_1+1}^{n} \left(RS_{E_{i_1}} \cap RS_{E_{i_2}} \right) \right) \cap AS. \tag{2.15}$$



(a) RS_{E_1} (b) RS_{E_2} (c) $RS_{E_1} \cup RS_{E_2} = T_1$

(d) T_1 (e) RS_{E_3} (f) $T_1 \cup RS_{E_3} = T_2$

(g) T_4 (h) RS_{E_6} (i) $T_4 \cup RS_{E_6} = RS_{TE}$

(j) RS_{TE} (k) AS (l) $RS_{TE} \cap AS = C_{1\times}$

Figure 2.17: Single Coverage Illustration – 6 Satellite Constellation

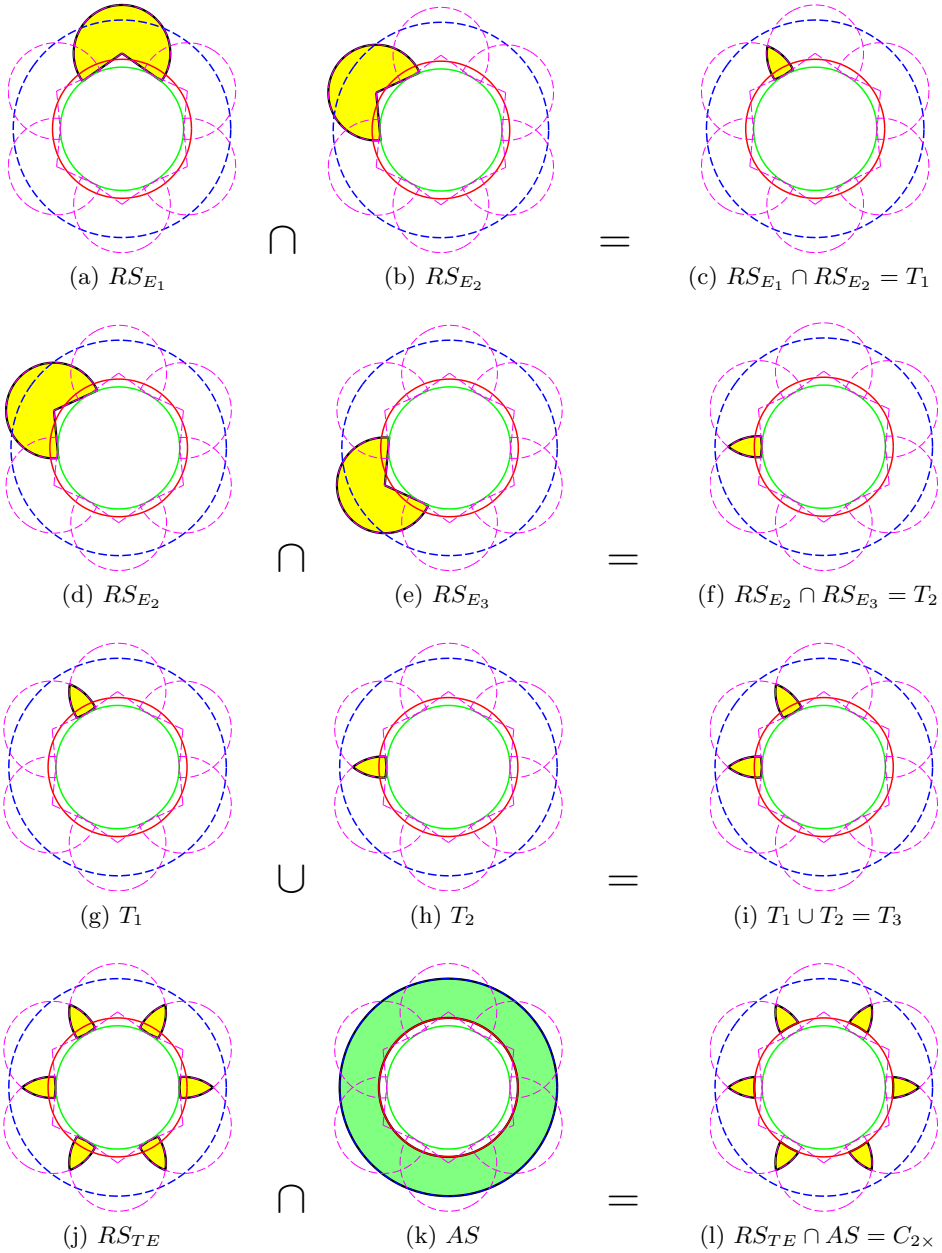
The values of the indices on the $\bigcup$ operators are chosen to avoid redundant or meaningless calculations. For instance, performing $RS_{E_1} \cap RS_{E_2}$ followed by $RS_{E_2} \cap RS_{E_1}$ is a redundancy (they represent the same region – the intersection operator is commutative). Similarly, considering $RS_{E_1} \cap RS_{E_1}$ for double coverage is meaningless – RS_{E_1} cannot cover the same region twice. The enclosed area of $C_{2\times}$ is evaluated using the method given in Equation 2.14.

2.3.3 Triple Coverage

The region of triple coverage is determined in similar fashion to the region of double coverage. Instead of a single intersection operation between pairs, as in the double coverage case, two intersection operations are carried out to find the region of triple coverage between a triplet of satellites. This process is illustrated in Figure 2.19. Figures 2.19a-2.19f illustrate the determination of one region of triple coverage. Once regions of triple coverage are determined, they are joined together by a sequence of union operations, as shown in Figures 2.19g-2.19i, resulting in the total effective sensor region shown in Figure 2.19j. Finally, as before, an intersection operation is performed between the total effective sensor region and the altitude shell, yielding the region of triple coverage within the region of interest, given by $C_{3\times}$ in Figure 2.19l. This region can be expressed in set notation as

$$C_{3\times} = \left(\bigcup_{i_1=1}^{n-2} \bigcup_{i_2=i_1+1}^{n-1} \bigcup_{i_3=i_2+1}^{n} \left(RS_{E_{i_1}} \cap RS_{E_{i_2}} \cap RS_{Ei_3} \right) \right) \cap AS. \tag{2.16}$$

As before, the area enclosed by $C_{3\times}$ is numerically evaluated using the relation shown in Equation 2.14.



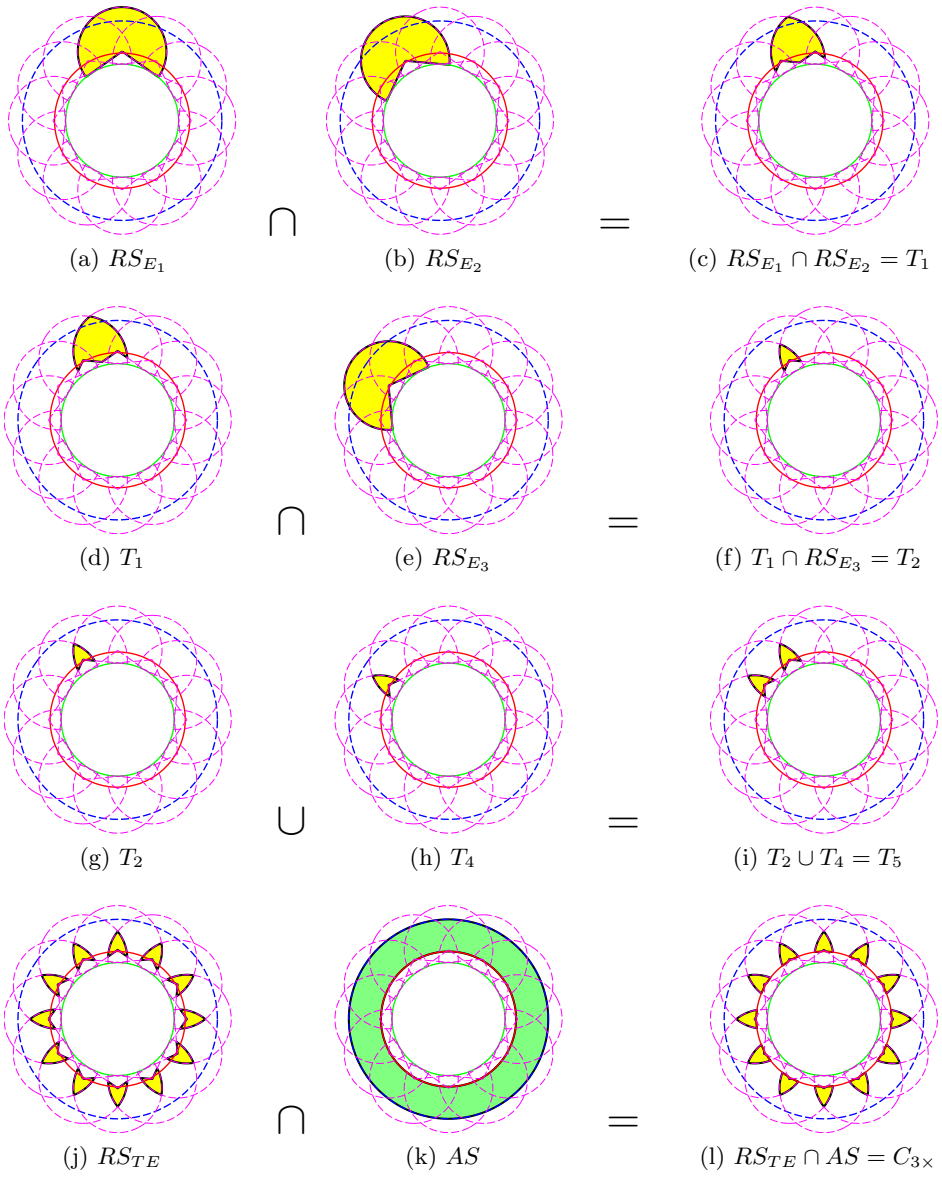
(a) RS_{E_1} (b) RS_{E_2} (c) $RS_{E_1} \cap RS_{E_2} = T_1$

(d) RS_{E_2} (e) RS_{E_3} (f) $RS_{E_2} \cap RS_{E_3} = T_2$

(g) T_1 (h) T_2 (i) $T_1 \cup T_2 = T_3$

(j) RS_{TE} (k) AS (l) $RS_{TE} \cap AS = C_{2\times}$

Figure 2.18: Double Coverage Illustration – 6 Satellite Constellation



(a) RS_{E_1} (b) RS_{E_2} (c) $RS_{E_1} \cap RS_{E_2} = T_1$

(d) T_1 (e) RS_{E_3} (f) $T_1 \cap RS_{E_3} = T_2$

(g) T_2 (h) T_4 (i) $T_2 \cup T_4 = T_5$

(j) RS_{TE} (k) AS (l) $RS_{TE} \cap AS = C_{3\times}$

Figure 2.19: Triple Coverage Illustration – 12 Satellite Constellation

2.3.4 Arbitrary Coverage Multiplicity

Considering the single, double, and triple coverage models individually allows for the discovery of a pattern in the set notation expressions. It then becomes clear (and is easily verified by hand, simply by enumerating all possible unique p-tuplet sets) that the coverage region of an arbitrary p multiplicity (where p is an integer) of an n satellite constellation ($C_{p\times}$) is expressed in set notation as

$$C_{p\times} = \left(\bigcup_{i_1=1}^{n-p+1} \bigcup_{i_2=i_1+1}^{n-p+2} \ldots \bigcup_{i_{p-1}=i_{p-2}+1}^{n-1} \bigcup_{i_p=i_{p-1}+1}^{n} \bigcap_{j=1}^{p} RS_{E_{i_j}} \right) \cap AS. \tag{2.17}$$

Although it is implied in the equation above, it is necessary to point out that $n \geq p$. This is clear when considering the opposite case by logic alone, regardless of the equation – for example, it is impossible for a three satellite constellation to yield quadruple coverage. Note that the expression reduces to the previously discussed single, double, and triple coverage models for p values of 1, 2, and 3 respectively.

As with the single, double, and triple coverage models, the enclosed area of $C_{p\times}$ is evaluated using Equation 2.14.

2.3.5 Number of Unique p-tuple Sets

To determine an upper bound on the number of clip operations per coverage model evaluation, it is helpful to count the possible number of intersection operation terms that result from the total number of finite unions in Equations 2.13, 2.15, 2.16, and 2.17. Let $q_p(n)$ be an integer for the p-multiplicity case giving the number of unique p-tuple sets among n satellites. The sets must be 'unique' in that there are no duplications by permutation (i.e. for $p = 3$,

comparing satellite sensor regions $(RS_{E_5}, RS_{E_6}, RS_{E_7})$ is equivalent to comparing regions $(RS_{E_7}, RS_{E_6}, RS_{E_5})$), and self comparisons, i.e. attempting to analyze for triple coverage between a satellite sensor region and itself, as in the cases of $(RS_{E_1}, RS_{E_1}, RS_{E_1})$ or $(RS_{E_1}, RS_{E_1}, RS_{E_3})$ – each satellite sensor region is only considered to cover a region once.

Considering single coverage, the unique sets are composed of single satellites – only one observing satellite is necessary for a region of single coverage. These single satellite sets are their own single-coverage regions, and require no intersection operations to determine them. Thus, the total number of unique sets $q_1(n)$ is simply equal to n.

For double coverage, the expression $q_2(n)$ is determined quite intuitively if a suitable analog is constructed. Consider the n satellite case – let the i-th effective satellite range shell, RS_{E_i}, be denoted by its index, i, for brevity. All possible satellite range shell pairings are arranged into an array as

$$\begin{pmatrix} (1,1) & (1,2) & (1,3) & \cdots & (1,n) \\ (2,1) & (2,2) & (2,3) & \cdots & (2,n) \\ (3,1) & (3,2) & (3,3) & \cdots & (3,n) \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ (n,1) & (n,2) & (n,3) & \cdots & (n,n) \end{pmatrix}. \tag{2.18}$$

The above array contains n^2 elements, however, clearly the n entries on the diagonal are invalid because they prescribe comparisons searching for double coverage regions between a satellite and itself (as mentioned before, physically impossible). Omitting

these n diagonal elements leaves $n^2 - n$ elements in the array as

$$\begin{pmatrix} \cancel{(1,1)} & (1,2) & (1,3) & \cdots & (1,n) \\ (2,1) & \cancel{(2,2)} & (2,3) & \cdots & (2,n) \\ (3,1) & (3,2) & \cancel{(3,3)} & \cdots & (3,n) \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ (n,1) & (n,2) & (n,3) & \cdots & \cancel{(n,n)} \end{pmatrix}. \tag{2.19}$$

Next, by noting the commutativity of the intersection operations performed on each pair (i.e. $RS_{E_1} \cap RS_{E_2} = RS_{E_2} \cap RS_{E_1}$), the lower triangular portion of the matrix can be rewritten as

$$\begin{pmatrix} \cancel{(1,1)} & (1,2) & (1,3) & \cdots & (1,n) \\ (1,2) & \cancel{(2,2)} & (2,3) & \cdots & (2,n) \\ (1,3) & (2,3) & \cancel{(3,3)} & \cdots & (3,n) \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ (1,n) & (2,n) & (3,n) & \cdots & \cancel{(n,n)} \end{pmatrix}, \tag{2.20}$$

which is clearly a symmetric array. Because duplicate entries are omitted, only half of the remaining entries are necessary, leaving a total of $(n^2 - n)/2$ unique pairs of satellites, or

$$\begin{pmatrix} \cancel{(1,1)} & (1,2) & (1,3) & \cdots & (1,n) \\ \cancel{(1,2)} & \cancel{(2,2)} & (2,3) & \cdots & (2,n) \\ \cancel{(1,3)} & \cancel{(2,3)} & \cancel{(3,3)} & \cdots & (3,n) \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ \cancel{(1,n)} & \cancel{(2,n)} & \cancel{(3,n)} & \cdots & \cancel{(n,n)} \end{pmatrix}. \tag{2.21}$$

Thus, $q_2(n) = (n^2 - n)/2$. While a similar analysis could be performed for higher coverage multiplicity cases, the method used above to derive $q_2(n)$ is no longer intuitive. This is especially true for coverage multiplicities of $p > 3$, where easily imagined spatial arrangements of the possible sets do not exist.

As an alternative to such non-intuitive derivations, a MATLAB script (iterating upon the indices defined in Equation 2.17) is used to count the number of

unique p-tuplets over all values of n (between p and $2p$) for each value of p between 3 and 8. It is clear, by intuition gleaned from investigations into the single, double, and triple coverage cases, that the highest degree term in each q_p expression is of degree p. Thus, a p-degree polynomial curve fit is applied to each data set to determine coefficients. The decimal coefficients are found to correspond (to very high precision) to the rational number coefficients shown in the following expressions:

$$q_1 = n, \tag{2.22}$$

$$q_2 = \frac{n^2 - n}{2}, \tag{2.23}$$

$$q_3 = \frac{n^3 - 3n^2 + 2n}{6}, \tag{2.24}$$

$$q_4 = \frac{n^4 - 6n^3 + 11n^2 - 6n}{24}, \tag{2.25}$$

$$q_5 = \frac{n^5 - 10n^4 + 35n^3 - 50n^2 + 24n}{120}, \tag{2.26}$$

$$q_6 = \frac{n^6 - 15n^5 + 85n^4 - 225n^3 + 274n^2 - 120n}{720}, \tag{2.27}$$

$$q_7 = \frac{n^7 - 21n^6 + 175n^5 - 735n^4 + 1624n^3 - 1764n^2 + 720n}{5040}, \tag{2.28}$$

$$q_8 = \frac{n^8 - 28n^7 + 322n^6 - 1960n^5 + 6769n^4 - 13132n^3 + 13068n^2 - 5040n}{40320}. \tag{2.29}$$

2.3.6 Number of Clip Operations

Having established the number of unique p-tuple sets for a given coverage multiplicity in Section 2.3.5, an upper bound on the number of clip operations required for each function evaluation is then determined. Performing clip operations consumes the vast majority of computer runtime during a coverage model evaluation, thus, establishing these bounds is valuable for estimating program runtime.

The single coverage model, stated in Equation 2.13, first requires $n - 1$

union operations to join the n satellite range shells, followed by a single intersection operation with the altitude shell to form $C_{1\times}$, bringing the total number of clip operations to n – the same as $q_1(n)$.

The double coverage case, defined in Equation 2.15, is slightly more complex. For each of the $q_2(n)$ unique pairs of satellites compared, one intersection operation is performed. The resulting individual regions of all intersection operations are then joined together in $q_2(n) - 1$ union operations, followed by one intersection operation with the altitude shell to form $C_{2\times}$. Thus, the total number of clip operations is $q_2(n) + q_2(n) - 1 + 1 = 2q_2(n)$.

Similarly, for the triple coverage case, defined in Equation 2.16, each of the $q_3(n)$ unique triplets requires two intersection operations to fully process. The resulting $q_3(n)$ regions of triple coverage require up to $q_3(n) - 1$ union operations to form the total triple coverage region. This is then followed by one intersection operation with the altitude shell to finally form $C_{3\times}$. The total number of clip operations is then $2q_3(n) + q_3(n) - 1 + 1 = 3q_3(n)$.

The same principle applies to higher coverage multiplicities as well. For p-multiplicity coverage, there are $q_p(n)$ p-tuplets, that each require $p - 1$ intersection operations to fully process. The resulting $q_p(n)$ regions of p-multiplicity coverage require $q_p(n) - 1$ union operations to form the total p-multiplicity coverage region. Just as before, a final intersection operation is performed with the altitude shell to produce $C_{p\times}$. Thus, the total number of clip operations is $(p-1)q_p(n)+q_p(n)-1+1$, which simplifies to

$$Q_p(n) \equiv pq_p(n). \tag{2.30}$$

While $Q_p(n)$ forms an upper bound for the number of clip operations, it will almost never be reached in a practical coverage model implementation. Consider the case shown in Figure 2.20, where $n = 12$ and triple coverage is considered. Note that the satellites (the icons on top of the shaded arrow-head shaped triple coverage regions) are numbered position-wise according to the digits on a standard analog clock for convenience. The total list of unique triplets that must be analyzed to find all triple coverage regions contains $q_3(12) = 220$ triplets. Thus, a comprehensive list is omitted for brevity, and instead, a small set of example triplets is shown in Table 2.2.

Table 2.2: Example Triplets – $p = 3$, $n = 12$ (see Fig. 2.20)

Set	Exists?	Comment
$(1, 2, 3)$	Y	–
$(1, 2, 8)$	N	$RS_{E_1} \cap RS_{E_8}$ and $RS_{E_2} \cap RS_{E_8}$ DNE
$(1, 7, 8)$	N	$RS_{E_1} \cap RS_{E_7}$ and $RS_{E_1} \cap RS_{E_8}$ DNE
$(4, 5, 10)$	N	$RS_{E_4} \cap RS_{E_{10}}$ and $RS_{E_5} \cap RS_{E_{10}}$ DNE
$(6, 9, 12)$	N	no overlap at all
$(11, 12, 1)$	Y	–

Because the p range shells in each p-tuplet must be analyzed with $p - 1$ intersection operations, it is common that an empty region is produced before all $p-1$ intersection operations have been completed. Once the intermediate region is null, all future intersection operations involving it will also be null. Thus, the algorithm can move immediately to processing the next p-tuplet avoiding several unnecessary clip operations. Additionally, any empty regions resulting from a p-tuplet with no coverage will be omitted from the union process combining regions of desired coverage multiplicity. A thorough implementation could also be optimized by ensuring

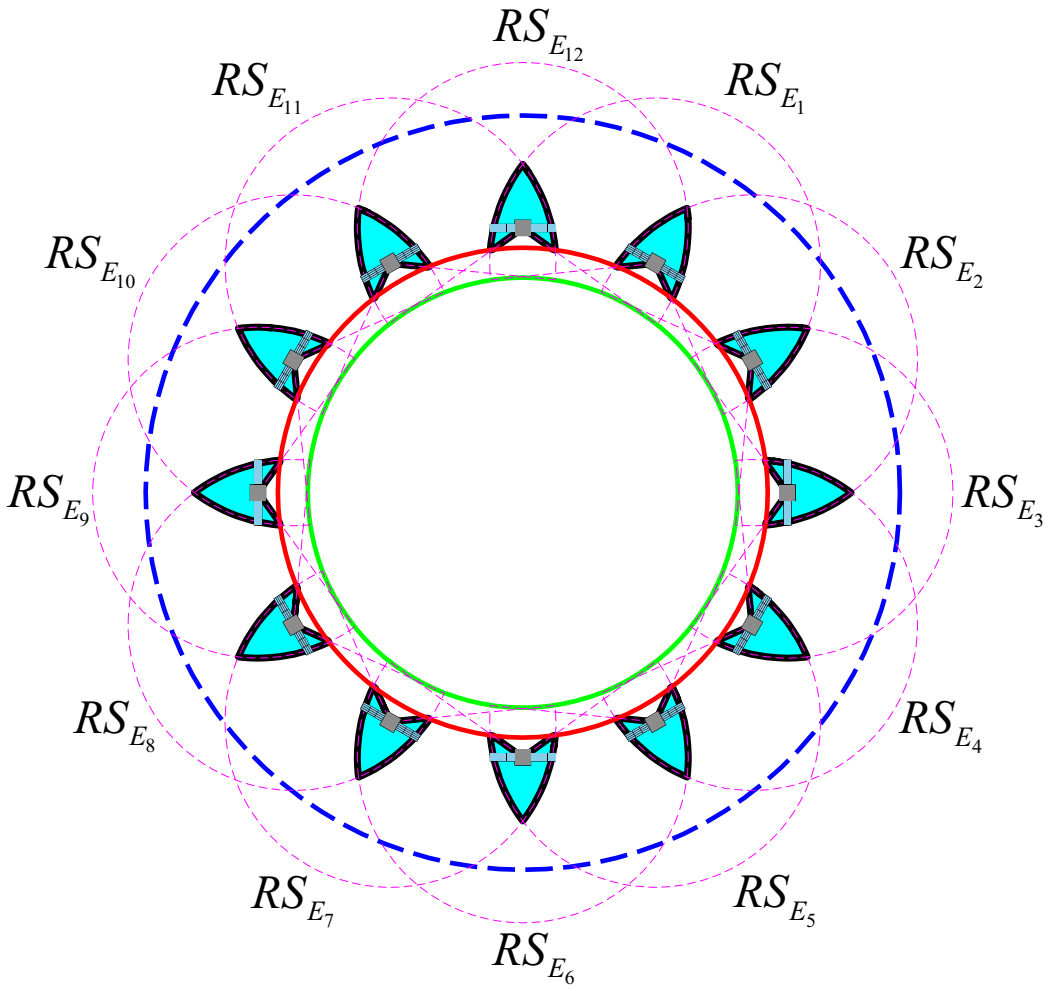


Figure 2.20: Triple Coverage Illustration

that future p-tuplets containing sensor regions known to be non-comparable are skipped altogether, maintaining what amounts to a 'blacklist' of satellite pairings.

Chapter 3

Implementation and Validation

In this chapter the implementation used to numerically evaluate ATH coverage is explored in detail. First, the various polygon clipping implementations used in this study are discussed. Performance comparisons are presented, revealing that the C++ implementation using the General Polygon Clipping (GPC) library[6] provides the best performance. Next, the implementation of the numerical ATH coverage models discussed in Section 2.3 is described in pseudo-code, along with a brief example algorithm carrying out an analysis of ATH coverage at multiple coverage multiplicities. Finally, the coverage model implementations are explored in detail, characterizing their approximation error and performance.

3.1 Clipping Implementations

Following the derivation and implementation of the analytical ATH coverage model presented in Appendix A, the next logical step is to develop a numerical implementation, which is readily validated using the analytical coverage model. The numerical approach used in this study defines the various in-plane regions as polygons for which manipulation algorithms already exist (this is a frequent operation in computer graphics). The methods used in this approach are discussed extensively in Chapter 2. Preliminary investigations focus on numerically reproducing

the analytical results obtained by Marchand and Kobel[4] and discussed in Section 1.1.2.

3.1.1 General Polygon Clipping Library

The General Polygon Clipping (GPC) library, developed and maintained by Alan Murta,[6] is a robust C implementation of Vatti's polygon clipping algorithm.[17] Vatti's algorithm is discussed in Section 2.2.3.1. GPC is freely available for non-commercial purposes, and has achieved widespread use and implementation in both industry and academia.

For example, the first MATLAB clipping implementation used in this research (developed by Jacquenot),[3] described in Section 3.1.3, relies on GPC via a MATLAB Executable (MEX) interface written by Hölz.[7] Additionally, MathWorks' MATLAB Mapping Toolbox (unavailable for regular use by the investigator) features an implementation of GPC via their own MEX gateway in a function named *polybool*.[5] The robustness and convenient interface of the GPC library has made it a benchmark for testing new polygon clipping algorithms in computational geometry publications. Consequently, the present investigation focuses on GPC-based codes for all polygon clipping implementations.

3.1.2 Martínez Polygon Clipping Algorithm Implementation

In the course of this study, the clipping algorithm developed by Martínez[2] (Section 2.2.3.3), is partially implemented in MATLAB by the investigator. As the algorithm is new, there does not yet exist a freely available implementation with

proven robustness at the time of this research. Based upon analysis presented by Martínez, his algorithm should provide as much as an order of magnitude improvement over GPC[6] for the polygon resolutions used in this study.

Although this implementation functions correctly for the vast majority of situations, it still lacks the proper facilities for handling all degenerate cases. Degenerate cases, identified by collinear or nearly collinear line segments and certain cases of coincident vertices, are difficult to address, and are the primary obstacle to overcome in developing a robust implementation. It is for this reason that nearly all computational geometry textbooks fail to address the issue, frequently citing that it is beyond the scope of the presented material.

Although a fully robust version of the implementation has not yet been realized, the development process itself has been very instructive in developing intuition and knowledge of how polygon clipping algorithms work. This knowledge has proved invaluable for troubleshooting while implementing the numerical ATH coverage models discussed in Section 2.3.4.

Additionally, because the code is written entirely in MATLAB (a C++ port is planned, pending completion of MATLAB development), its poor performance illustrates why there are no existing polygon clipping codes that operate natively in MATLAB. Martínez' algorithm[2] is most readily implemented using an object-oriented approach. However, because MATLAB interprets code as it runs, as opposed to executing pre-compiled binaries, the object-oriented code proves to be exceptionally slow.

Object creation and destruction alone are found to be a significant perfor-

mance bottleneck in MATLAB. This is a problem considering that a given clipping problem with a combined total of n vertices requires at least n line segment objects, and $2n$ line segment endpoint objects, that must be created and subsequently destroyed during each clipping operation. In contrast, this object management bottleneck is not observed while using object-oriented code in C++.

The performance of the MATLAB implementation of the Martínez algorithm is so poor its analysis is excluded from the comparison of clipping implementation performance, given in Section 3.1.6. For a 100 PPC polygon resolution, the Martínez MATLAB implementation solves problems at approximately 5 seconds per function evaluation – four orders of magnitude longer than the most efficient implementations shown in Table 3.2. A full 9900 data point comparison analysis, as is used for the data in Table 3.2, is impractical at higher polygon resolutions due to time constraints.

Thus, despite expectations of performance improvements in C++, the poor MATLAB performance of the code was a hindrance to early research efforts. In addition to this, the lack of support for certain degenerate cases sometimes causes a long-running analysis to crash. By setting this implementation aside for the time being, a great deal of progress has been made using GPC-based codes.[6]

For future studies, completion of the Martínez[2] implementation in C++ may yield a significant performance improvement. This should be explored before attempting to solve larger design problems with long expected solution times.

3.1.3 Preliminary Implementation – *Polygon Intersection* code

A MATLAB implementation using Vatti's algorithm,[17] known as *Polygon Intersection*[3] by Jacquenot, was obtained from the MathWorks File Exchange. Jacquenot's code is essentially a wrapper for a GPC-based[6] MATLAB Executable (MEX) developed by Hölz[7] – this code is discussed further in Section 3.1.4. *Polygon Intersection* uses a MATLAB cell array of data structures defining polygon contours for both input and output purposes. The function is capable of identifying regions of intersection between any number of these polygons.

To verify that the code functions as expected, a simple test case is created and analyzed – one where the resulting area is easily verified by hand. A rectangular polygon with a hole, and an L-shaped polygon are defined and supplied to the *Polygon Intersection* code. The areas are easily calculated by hand, and exactly match the program output, preliminarily demonstrating proper code function. This test case is shown in Figure 3.1 (drawn using Jacquenot's supplied plotting method).

Constructing an *ad hoc* numerical ATH coverage model to mimic the output of the analytical coverage model (documented in Appendix A) is straightforward as the definition of polygons and the associated operations performed on each are easily identified. A single polygon represents the effective range shell, as described in Section 2.1.3.1, and a polygon in the shape of an annulus defines the dual-altitude band shell, as discussed in Section 2.1.3.3. Vertex coordinates describing these two polygons at a resolution of 100 PPC are added to the correct input data structure fields, and the *Polygon Intersection* code returns a data structure describing the resulting regions as well as indices describing which polygons overlap in which

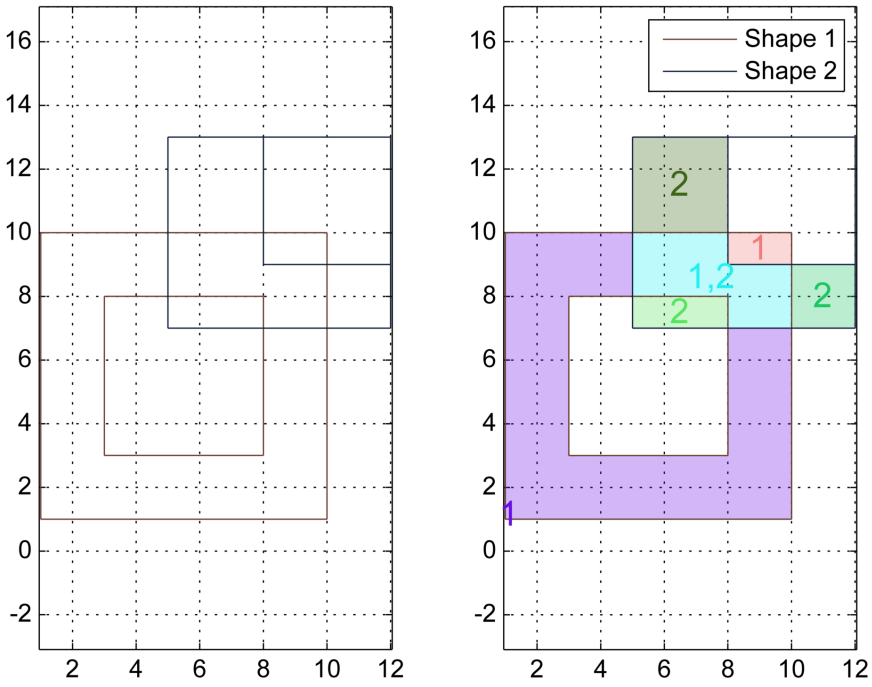


Figure 3.1: Test Case Using *Polygon Intersection*[3]

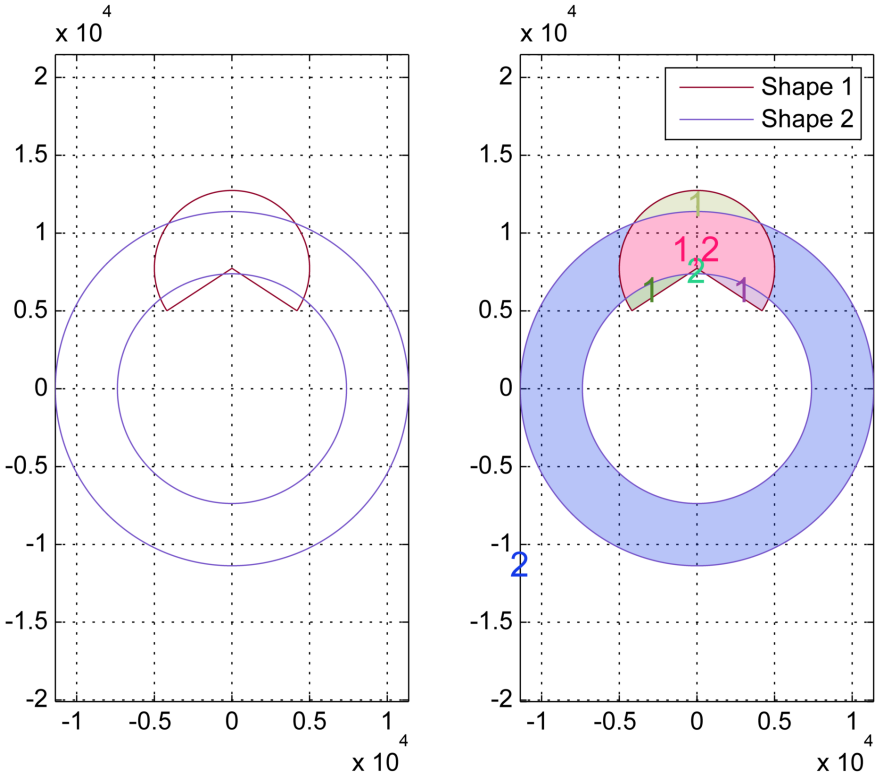


Figure 3.2: 1× Coverage Region (Labeled '1,2') Found Using *Polygon Intersection*[3]

regions.

From these indices it is straightforward to determine the region of intersection covered by both the effective range shell and altitude shell. The resulting regions (illustrated using Jacquenot's[3] provided methods) are shown in Figure 3.2. The coverage region of interest is denoted by the $(1, 2)$ labels, where regions 1 (effective range shell) and 2 (dual-altitude band shell) overlap.

The associated areas of each region are computed using the *polyarea*[26] MATLAB function and added as a field to the output data structure. For the situation

shown in Figure 3.2, the code numerically computes the coverage area to within the expected deviation (less than 0.1% relative error compared to the analytically obtained value), as discussed in Section 2.1.2.1.

3.1.4 MATLAB Implementation – GPC via Direct MEX Gateway

Based upon observations made while developing the Martínez[2] algorithm implementation, it was considered likely that any MATLAB data sorting and processing involved in the polygon clipping process would be detrimental to performance. This supposition turns out to be correct, as will be shown in Section 3.1.6. By directly interfacing with the MEX gateway, and only performing necessary operations, a great performance increase is observed relative to both Jacquenot's *Polygon Intersection* code,[3] and MathWorks' own *polybool* function[5] in their Mapping Toolbox. Both of these implementations access GPC via a MEX interface, but they both carry out a large amount of data processing tasks on the MATLAB side, introducing a significant amount of overhead.

The MEX interface used by Jacquenot[3] was developed by Hölz,[7] and is available on the MathWorks File Exchange. As for MathWorks' *polybool* code,[5] inspection of the associated MATLAB m-code makes it quite clear that GPC is in use via MEX. Presumably, MathWorks is using their own proprietary MEX interface to GPC; the source code is consequently unavailable to the investigator.

For the most direct implementation of GPC[6] from MATLAB, Hölz' GPC MEX interface[7] is used in the current study. From inspecting the MEX source code (in C), it is clear that his interface was developed with a minimalist approach, and

does not introduce any avoidable bottlenecks to performance.

3.1.5 C++ Implementation – Modified GPC

GPC[6] is a native C library, and thus, incorporating it into a C++ program is straightforward. An analysis program running entirely in C/C++ offers superior performance to even the adequately fast MATLAB GPC/MEX[7] implementation discussed in Section 3.1.4. Compiled C/C++ binaries have more direct access to computer system resources than programs running on an interpreted platform such as MATLAB.

In the MATLAB implementation discussed in Section 3.1.4, polygon structure copy operations are handled on the MATLAB side, where memory allocation and deallocation are managed automatically. However, C++ does not feature this type of memory management, and allocation and deallocation must be performed explicitly (an exception to this is when using many of the built-in C++ derived data types).

Problems were initially encountered when attempting to copy polygon data between intermediate states (as is discussed in Section 3.2.2), resulting in memory leaks.[27] These memory leaks were attributed to a failure to deallocate old polygon data before reallocating space for new data. The presence of this leak was first noted during a long-running analysis when a nearly linear increase over time in memory use by the program was observed.

In order to resolve this issue, and maintain adherence to the general structure of the algorithm developed in the MATLAB implementation, a polygon copy

method is implemented, where correct deallocation and allocation is performed on the polygon structure receiving the new data. When this new method is properly used, memory usage of the program remains bounded over time.

Additionally, a very simple method is developed to initialize empty polygon structures of the GPC polygon type. This method explicitly initializes polygon structure fields (that would otherwise be uninitialized) to values that reflect that it is indeed empty. Without this step, determining if a polygon structure is empty or not before it is clipped will return erroneous results causing unpredictable program behavior.

3.1.6 Performance Comparison

Using the *ad hoc* single coverage model to numerically mimic the published analytical results by Marchand and Kobel,[4] it is desired to compare the performance of these first numerical implementations against the analytical coverage model, shown in Appendix A.

The problem parameters, summarized in Table 3.1, are taken directly from Marchand and Kobel's published results[4] (presented there as 'Example 1'). The published example depicts coverage area as a function of varying satellite altitude while keeping all other parameters fixed, i.e. producing the area of coverage vs. altitude plot shown in Figure 3.3.

This published example is selected for comparison because it has previously been used for validating the analytical coverage model implementation shown in Appendix A. The analytical results are compared against numerically obtained results

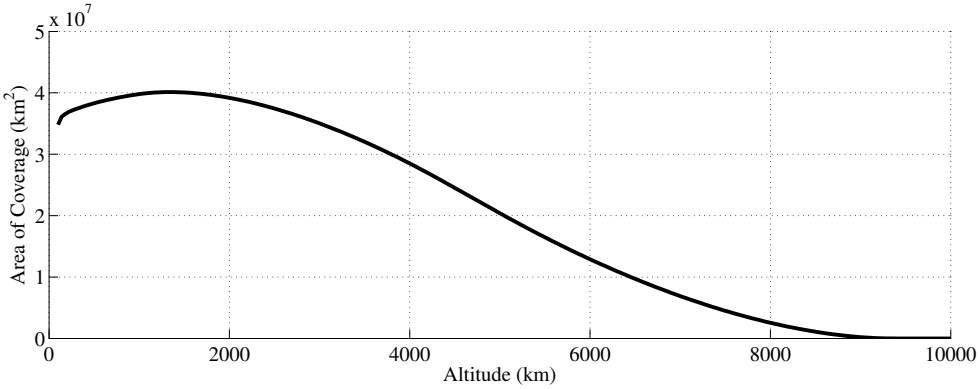


Figure 3.3: Area of Coverage vs. Altitude for a Single Satellite Using the Parameters in Table 3.1 (Marchand and Kobel's[4] 'Example 1')

computed using polygon resolutions of 100, 1000, and 4000 PPC. Polygon resolutions are restricted to this small set simply because several of the implementations being compared are so slow that more exhaustive studies are prohibitive due to the limited computer time available.

The platform used for these comparative analyses is a 2.0 GHz Intel Core 2 Duo running the MATLAB 7.10 Student Edition on 32-bit Windows XP. For the GPC/C++ implementation, the Microsoft Visual C++ 2010 compiler is used with all compiler optimization settings at their default values.

For each clipping implementation and at each polygon resolution, the entire regime is analyzed in 1 km increments, which, between $h_{s_{\min}} = 100$ km and $h_{s_{\max}} = 10000$ km, results in 9900 data points for each run.

Shortly before completion of the research presented in this thesis, the *polybool*[5] function from the MATLAB Mapping Toolbox was briefly borrowed in order

Table 3.1: Parameters for Analytical vs. Numerical Comparison

Parameter Value	Description
$n = 1$	number of satellites
$p = 1$	coverage multiplicity
$R_e = 6378.14$ km	assumed Earth radius
$R = 5000$ km	omni-directional sensor range
$h_t = 100$ km	tangent height
$h_l = 1000$ km	lower altitude bound
$h_u = 5000$ km	upper altitude bound
$h_{s_{\min}} = 100$ km	lower limit on satellite altitude
$h_{s_{\max}} = 10000$ km	upper limit on satellite altitude

to gauge its performance against the other implementations discussed here. The MATLAB function, as discussed before, relies upon GPC[6] via a (proprietary) MEX interface, and its performance is compared with the other implementations in Table 3.2.

Table 3.2: Average Runtimes – Analytical vs. Jacquenot,[3] *polybool*,[5] GPC/MEX,[6,7] and GPC/C++[6] at Varying Polygon Resolutions

Data Set	Avg. Runtime (s)	Avg. Clip time (ms)	Perf. Penalty
MATLAB Analytical	2.5	0.2525	–
Jacquenot 100 PPC	72.0	7.273	28.8×
Jacquenot 1000 PPC	250.7	25.32	100.3×
Jacquenot 4000 PPC	1207.3	121.9	482.9×
polybool 100 PPC	48.8	4.929	19.5×
polybool 1000 PPC	542.1	54.75	216.8×
polybool 4000 PPC	6281.9	634.5	2512.0×
GPC/MEX 100 PPC	11.8	1.192	4.7×
GPC/MEX 1000 PPC	67.2	6.788	26.9×
GPC/MEX 4000 PPC	455.8	46.04	182.3×
GPC/C++ 100 PPC	3.0	0.307	1.2×
GPC/C++ 1000 PPC	44.3	4.478	17.7×
GPC/C++ 4000 PPC	270.2	27.29	108.1×

As expected, a significant performance penalty is incurred when increasing polygon resolution for all implementations. The performance of the first code investigated (*Polygon Intersection*)[3] is so poor that it is immediately ruled out for future use. As discussed before, the poor performance of this code can be attributed to the operations being carried out in MATLAB. The direct GPC/MEX[6] implementation uses the exact same MEX interface (by Hölz),[7] but displays greater than an order of magnitude advantage in performance.

Perhaps the most unexpected result of all is the poor performance of *polybool*,[5] the function MathWorks includes in their MATLAB Mapping Toolbox. Also functioning by a GPC/MEX[6] interface, it is clear that either the MATLAB-side operations create a performance bottleneck, or their proprietary MEX interface is exceptionally inefficient.

Clearly from the results in Table 3.2, the GPC/C++[6] configuration is far superior to all others investigated. The GPC/C++ numerical implementation nearly achieves performance parity with the MATLAB analytical implementation for the 100 PPC case. This is quite an impressive result, when compared with the runtimes achieved using the other implementations shown in Table 3.2. The performance of the MATLAB GPC/MEX configuration, while not the best, is acceptable in that the MATLAB implementation is primarily used for development, small-scale analyses, and generating illustrations.

3.1.7 GPC/C++ Performance Analysis

Having established that the GPC/C++ clipping implementation offers the best performance, it is then useful to investigate more closely how its performance scales with polygon resolution. Consider the 'typical' circumstances for a single satellite coverage case, as illustrated in Figure 3.4. The parameters used are identical to those given in Table 3.1 with a satellite altitude of $h_s = 1349$ km. The selected value corresponds to the altitude of maximum single coverage obtained from reproducing the results of Marchand and Kobel.[4] This specific set of parameters results in an amount of polygon overlap frequently encountered in constellation configurations later in this study, thereby serving as an excellent basis for comparison. Excessive performance gains due to shortcuts are eliminated as the clipping implementation uses polygon bounding box data to reduce the time spent in non-comparable regions.

Figure 3.5 shows time per clip operation of the GPC/C++ clipping implementation for the intersection operation required to find single coverage of the configuration shown in Figure 3.4. This data is produced by performing clip operations continuously for at least 250 milliseconds at each polygon resolution from 10 PPC to 4000 PPC (a microsecond-resolution timing package is used). Polygon structures are allocated and deallocated before and after each operation to provide a realistic picture of overall implementation performance. The total time (slightly more than 250 milliseconds) is divided by the number of clip operations performed for each polygon resolution. Figure 3.5 shows a nearly linear relationship between polygon resolution and clip operation time.

An additional observation is that cases with odd-integer polygon resolutions

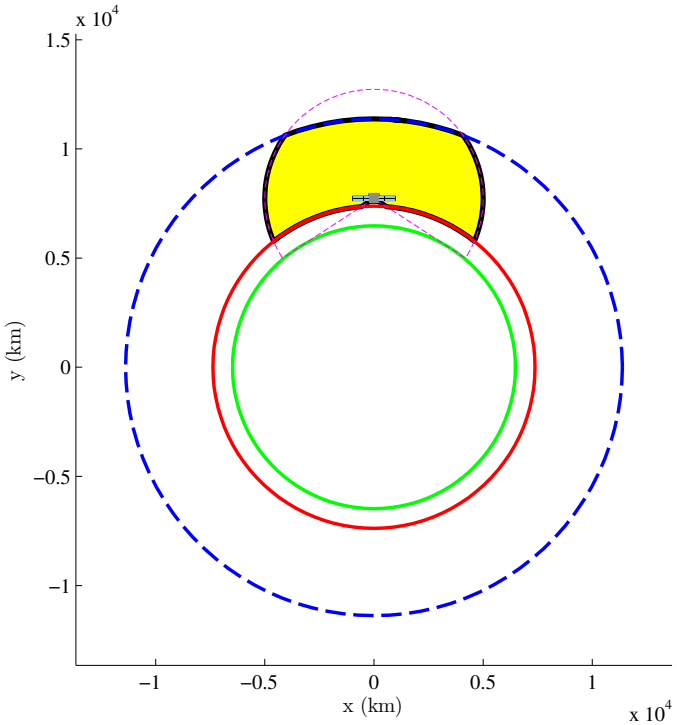


Figure 3.4: Satellite Configuration at the Altitude of Maximum Single Coverage, $h_s = 1349$ km (see Table 3.1, and Marchand and Kobel)[4]

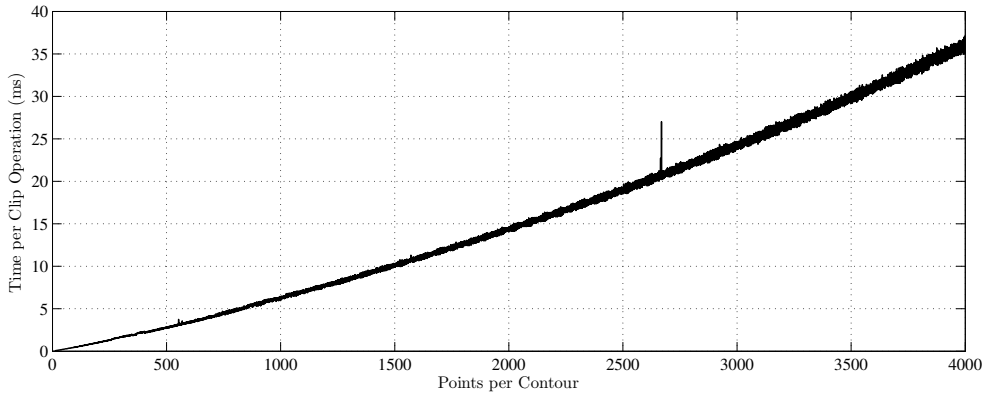


Figure 3.5: Time per Clip Operation (GPC/C++ Implementation) vs. PPC for the Configuration Shown in Figure 3.4

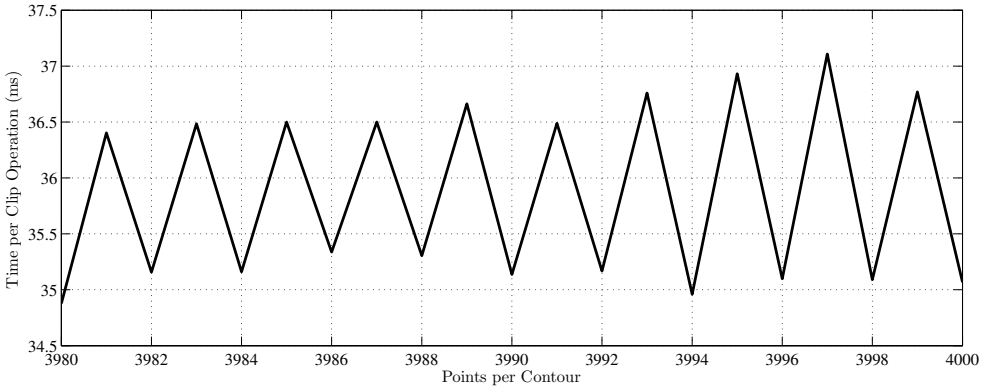


Figure 3.6: Detail From Figure 3.5 – 3980 to 4000 PPC

exhibit slightly slower performance than cases with even-integer resolutions (this occurs uniformly across the investigated regime), as shown in Figure 3.6. This behavior is responsible for the oscillation superimposed on the curve in Figure 3.5. A plausible cause for this variation could not be identified by the investigator.

Note that at the 1000 PPC data point, Figure 3.5 shows approximately 6.0 milliseconds per clip, whereas Table 3.2 finds 4.478 ms per clip at 1000 PPC. This discrepancy is a result of increased clipping efficiency for h_s values large enough to cause minimal polygon overlap (or none at all). In these circumstances, the clip time is substantially lower, and it serves to reduce the average clip time listed in Table 3.2. A similar argument can be made for the 100 and 4000 PPC data points.

To further investigate this behavior, the same problem (Marchand and Kobel's 'Example 1')[4] is solved again, this time analyzing time spent performing clip operations at different altitudes. Figures 3.7, 3.8, and 3.9 show the average time spent on each clip operation throughout the variation in satellite altitude (i.e. while

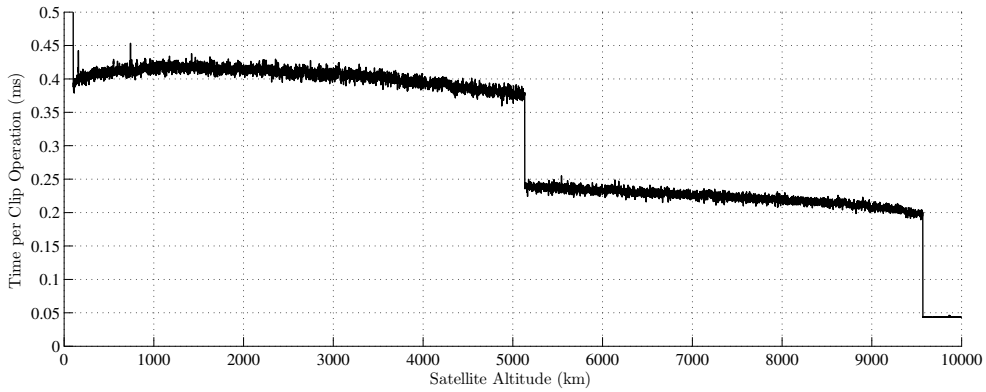


Figure 3.7: Time per Clip Operation vs. h_s – 100 PPC

numerically producing the curve shown in Figure 3.3). Aside from the discontinuous regions in the curve, clipping operations are carried out in nearly constant time.

It is important to note that the small transient spikes in clip time do not always appear in the same locations, or with the same frequency during subsequent runs of the exact same analysis. Because exactly the same calculations are performed from start to finish between runs, the discrepancy is attributed to system-related variations caused by programs competing for resources momentarily.

When comparing the results shown in Figures 3.7 through 3.9 with the altitude used to produce Figure 3.4 (1349 km), it is clear that the analysis shown in Figure 3.5 is conducted in the 'worst case' regime in terms of performance.

Finally, it is of interest to determine the conditions that precipitate the discontinuities in Figures 3.7 through 3.9. Altitudes before and after each discontinuity are selected, and their associated single coverage regions are illustrated in Figure 3.10.

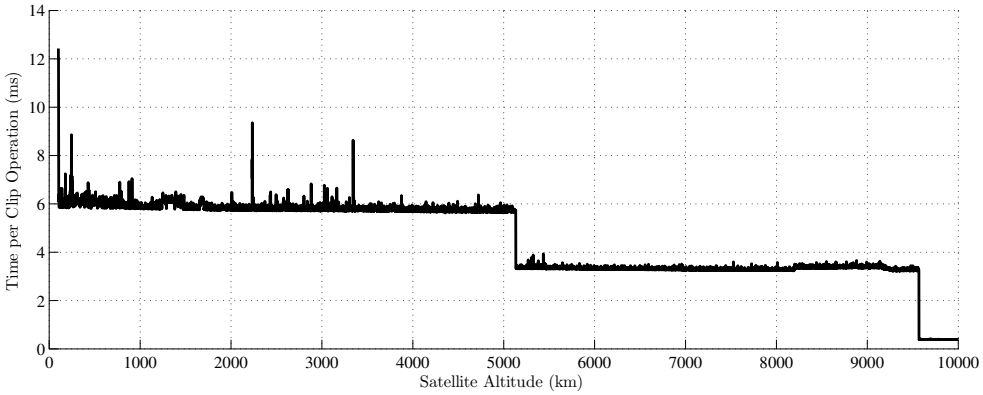


Figure 3.8: Time per Clip Operation vs. h_s – 1000 PPC

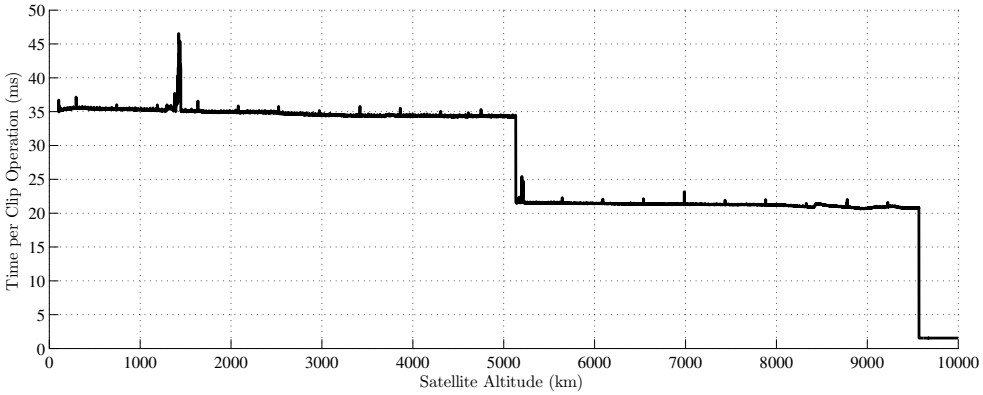
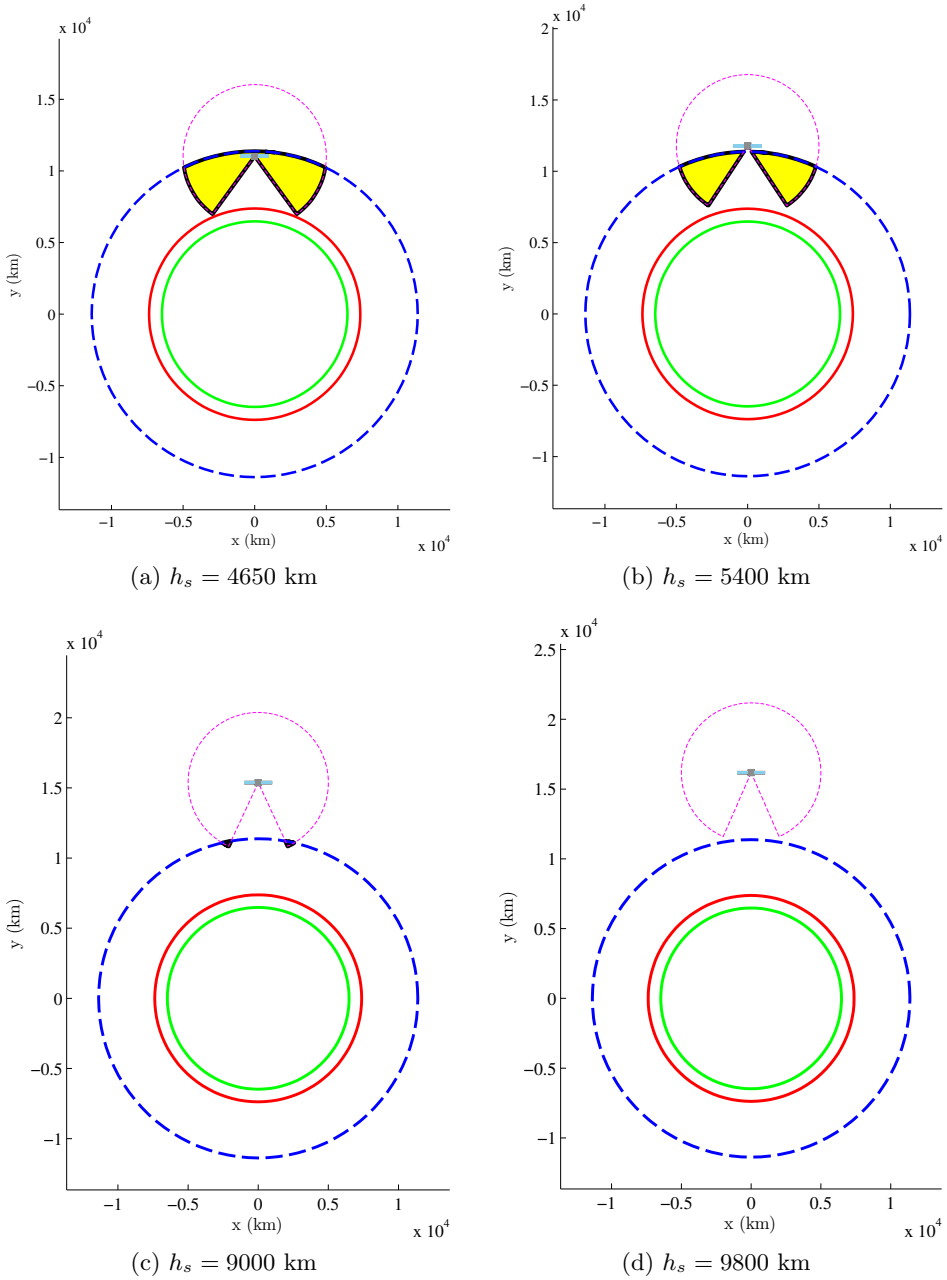


Figure 3.9: Time per Clip Operation vs. h_s – 4000 PPC



(a) $h_s = 4650$ km

(b) $h_s = 5400$ km

(c) $h_s = 9000$ km

(d) $h_s = 9800$ km

Figure 3.10: Configurations at Altitudes on Either Side of the Discontinuities in Figures 3.7, 3.8, and 3.9

Figures 3.10a and 3.10b show that the discontinuity near $h_s = 5000$ km in Figures 3.7 through 3.9 occurs when the coverage region bifurcates into separate and symmetric regions. This condition allows each region to be handled separately, resulting in an increase in performance. Figures 3.10c and 3.10d show the somewhat obvious situation, where the coverage regions become non-comparable, that is, their bounding boxes no longer overlap. In the case of an intersection operation, the polygon clipping implementation uses this condition to shortcut the clipping process and return an empty polygon structure. Alternatively, if the desired operation between two non-comparable polygons A and B is $A \cup B$, the clipping implementation simply returns the two polygon structures combined into one, whereas for $A - B$, A alone is returned.

3.2 Numerical ATH Coverage Model Implementation

The implementation of the numerical ATH coverage model comprises of three major parts. First, an array of effective range shells is generated – this allows a high degree of generality to be maintained in the rest of the code. Second, the region of total coverage at the desired multiplicity between the effective range shells is determined (itself a polygon region). Third, and finally, the resulting region of total coverage at the desired multiplicity is intersected with the region of interest (the dual-altitude band shell in this case). The area enclosed in this region corresponds to the in-plane area of ATH coverage at the desired coverage multiplicity.

Although regions of single, double, and triple coverage can be determined using the arbitrary coverage multiplicity implementation, given in Section 3.2.5,

they are treated separately as their algorithms are, unlike the arbitrary coverage case, an intuitive representation of the set notation discussed in Section 2.3.

3.2.1 Effective Range Shell Array, $\mathbf{RS}_E$

A common element of all coverage implementations in this section is the need to begin by generating and storing an $(n \times 1)$-dimensional array of polygon structures that describe the sensor coverage of each of the n satellites. This array is denoted by $\mathbf{RS}_E$. These polygon regions can be generated in any way, and can actually describe many different types of processes where the multiple overlap of polygons is of interest (i.e. not limited simply to the problem of ATH coverage).

One potential example in the field of geosciences is an analysis of polygons representing polar ice cap coverage of the Earth as recorded at weekly intervals. An analysis looking for $52\times$ coverage between those 52 polygons would yield the regions where there is *always* ice, while an analysis for $26\times$ coverage would reveal various regions where ice exists at least half of the year (although not necessarily continuously or contemporaneously with one another).

Once the array of effective range shells is generated, the coverage multiplicity algorithm described in Section 3.2.5 can compute the area of $p\times$ coverage between the n satellites. Sections 3.2.1.1 and 3.2.1.2 describe effective range shell array generation techniques for two possible circumstances (of many) that are applicable to the ATH coverage problem.

3.2.1.1 Omni-Directional Satellite Sensors, Circular Orbits

A case of particular interest to preliminary studies due to its simplicity and time-invariant nature is composed of n satellites equally spaced in longitude within a single circular orbit. This case is further simplified by assuming each satellite has omni-directional sensors. The sensor regions for each satellite are generated using the procedure discussed in Section 2.1.3.1. Algorithm 3.1 shows an example of this effective range shell array generation process. The input arguments are the tangent height shell (THS) radius, r_t, the circular satellite orbit radius, r_s, the omni-directional sensor range, R, the number of satellites, n, and the desired polygon resolution, m. The notation $\mathbf{RS}_E(i)$ indicates the i-th polygon structure, describing the effective range shell of the i-th satellite (out of a total of n) in the effective range shell array, $\mathbf{RS}_E$.

Algorithm 3.1 *GenerateRangeShellArray1*(r_t, r_s, R, n, m)

1: **for** $i = 1$ to n **do**
2: $\quad \theta_i = \frac{2\pi(i-1)}{n}$ {longitude in orbit of i-th satellite}
3: $\quad \mathbf{RS}_E(i) =$ *GenerateODRangeShell*$(r_t, r_s, R, \theta_i, m)$
4: **end for**
5: **return** $\mathbf{RS}_E$ {the effective range shell array}

The algorithm *GenerateODRangeShell* generates an effective range shell polygon for the omni-directional sensor case. This method is shown in Algorithm 3.2, and is an implementation of the procedure discussed in Section 2.1.3.1.

3.2.1.2 Arbitrary Sensor Profiles and Satellite Positions

More generally, suppose a constellation is composed of satellites featuring some arbitrary sensor profile for which the in-plane cross-section is known. The

Algorithm 3.2 *GenerateODRangeShell*(r_t, r_s, R, θ, m)

initialize RS_E {an empty polygon structure}
$\gamma = \arcsin(r_t/r_s)$
$\psi_i = \pi - \gamma + \theta$
$\psi_f = -\pi + \gamma + \theta$
$(x_1, y_1) = (r_s \cos\theta, r_s \sin\theta)$
add (x_1, y_1) to RS_E as the first vertex {centered on satellite}
for $j = 2$ to m **do**
    $\psi_j = \psi_i + \frac{(j-2)(\psi_f - \psi_i)}{m-2}$
    $x_j = R\cos\psi_j + x_1$
    $y_j = R\sin\psi_j + y_1$
    add (x_j, y_j) to RS_E as the j-th vertex
end for
return RS_E {omni-directional effective range shell for a single satellite}

cross-section is stored in a polygon structure RS_{arb} with the satellite at the origin of the coordinate frame RS_{arb} is defined within, as shown in Figure 3.11. The three-dimensional sensor profile is assumed to exhibit symmetry across the orbital-plane, as discussed in Section 2.1.3.2. Further, assume that the location and attitude of each satellite in the orbital plane is specified by a set of n (x, y) coordinate pairs and n angles. These are denoted by $n \times 2$ and n-dimensional vectors $\mathbf{r}_s$ and $\boldsymbol{\alpha}$, respectively. The attitudes and coordinates of each satellite can be defined in any desired two-dimensional coordinate system (can vary by application). This parameterization allows satellite positions to be specified by any number of means; i.e. time-varying analyses can be conducted by numerically integrating equations of motion, or from solving Kepler's Equation.[28] An outline of this procedure is shown in Algorithm 3.3. Note that, as discussed in Section 2.1.3.2, an additional clip operation is necessary to determine the region of the tangent height triangle (THT) that must be removed from the sensor cross-section to form the effective range shell.

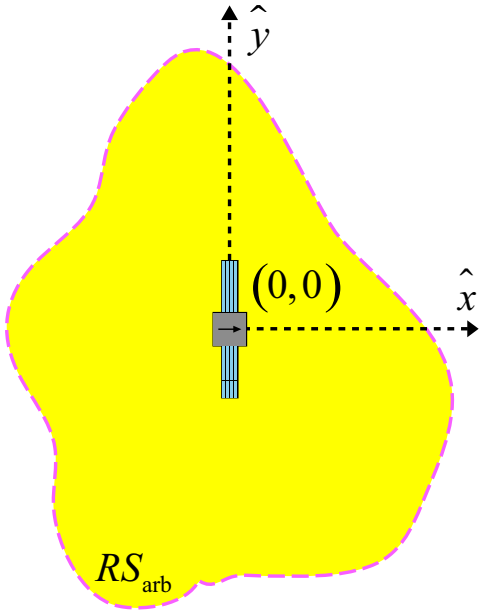


Figure 3.11: An In-Plane Cross-Section of an Arbitrary Sensor Profile RS_{arb}

Algorithm 3.3 *GenerateRangeShellArray2*$(\mathbf{r}_s, \boldsymbol{\alpha}, RS_{\text{arb}})$

1: $n =$ length of $\boldsymbol{\alpha}$
2: **for** $i = 1$ to n **do**
3: $\quad RS_{i_{\text{temp}}} = \mathbf{r}_s(i) + (RS_{\text{arb}}$ rotated by $\boldsymbol{\alpha}(i))$
4: $\quad$ initialize THT {a polygon structure defining the tangent height triangle}
5: $\quad \mathbf{RS}_E(i) = RS_{i_{\text{temp}}} -$ THT {using polygon clipping implementation}
6: **end for**
7: **return** $\mathbf{RS}_E$ {the effective range shell array}

This arbitrary range shell case is investigated in the example problem presented in Section 4.6, where the in-plane cross-section of an arbitrary sensor profile is defined by digitization of a hand drawing.

3.2.2 Regions of Single Coverage

The method of determining the region of single coverage is an extension of the *ad hoc* code used during preliminary investigation, discussed in Section 3.1. However, rather than initializing a single effective range shell polygon region, RS_E, an array of n effective range shell polygons, $\mathbf{RS}_E$, is initialized (as discussed in Section 3.2.1). Once initialized, these polygons are sequentially combined via $n-1$ union operations until the union of all effective range shells has been produced. The resulting polygon region, RS_{TE}, represents the total region of single coverage. In order to determine the region of single coverage within the area of interest (dual-altitude band shell, AS), an additional clipping operation is performed, as denoted by $RS_{TE} \cap AS$. This last step is discussed separately in Section 3.2.6.

Algorithm 3.4 shows the process by which the total region of single coverage is determined.

Algorithm 3.4 *SingleCoverage*($\mathbf{RS}_E$)

1: n = length of $\mathbf{RS}_E$
2: initialize $RS_{TE} = \mathbf{RS}_E(1)$ {the total effective range shell polygon}
3: **for** $i = 2$ to n **do**
4: $\quad RS_{TE_{\text{temp}}} = RS_{TE} \cup \mathbf{RS}_E(i)$
5: $\quad RS_{TE} = RS_{TE_{\text{temp}}}$
6: **end for**
7: **return** RS_{TE} {the total effective range shell polygon ($1\times$ coverage)}

Observe (lines 4, 5) that a temporary polygon structure, $RS_{TE_{\text{temp}}}$ is incorporated, and the union operation is split into two statements (as opposed to simply performing $RS_{TE} = RS_{TE} \cup \mathbf{RS}_E(i)$). This is performed (in all algorithms) because efficient polygon clipping implementations perform operations in-place. That is, polygons are passed-by-reference. Rather than create copies of the polygons for the clipping implementation to operate upon, only the memory addresses to the existing polygon structures are passed, and the polygon clipping implementation performs operations directly upon original copies in memory.

Such an approach minimizes the time and computational resource burden of allocating and deallocating copies of both input polygons at each function call. As a consequence of the plane-sweep method in use by the clipping algorithm (Section 2.2), as RS_{TE} and $\mathbf{RS}_E(i)$ are joined by union, $RS_{TE_{\text{temp}}}$ is constructed simultaneously. If $RS_{TE_{\text{temp}}}$ was equivalent to RS_{TE}, the clipping implementation would produce an erroneous result, or potentially crash entirely because an input polygon would be changing to reflect the partially constructed output polygon.

3.2.3 Regions of Double Coverage

The method of finding the total region of double coverage arises from an expansion of the single coverage method, as discussed in Section 3.2.2. An additional nested loop determines the intersection between members of each of the $q_2(n)$ pairs. Similarly to the single coverage method, $q_2(n) - 1$ union operations combine the resulting regions of double coverage (discussed in Section 2.3.6). Algorithm 3.5 illustrates the process for determining the total region of double coverage.

Algorithm 3.5 *DoubleCoverage*($\mathbf{RS}_E$)

1: n = length of $\mathbf{RS}_E$
2: **if** $n < 2$ **then**
3: error, necessary condition: $n \geq 2$
4: **end if**
5: initialize RS_{TE} {an empty polygon structure}
6: **for** $i_1 = 1$ to $n - 1$ **do**
7: **for** $i_2 = i_1 + 1$ to n **do**
8: $RS_{E_{i_1 i_2}} = \mathbf{RS}_E(i_1) \cap \mathbf{RS}_E(i_2)$
9: **if** RS_{TE} is empty **then**
10: $RS_{TE} = RS_{E_{i_1 i_2}}$
11: **else if** $RS_{E_{i_1 i_2}}$ is empty **then**
12: do nothing
13: **else**
14: $RS_{TE_{\text{temp}}} = RS_{TE} \cup RS_{E_{i_1 i_2}}$
15: $RS_{TE} = RS_{TE_{\text{temp}}}$
16: **end if**
17: **end for**
18: **end for**
19: **return** RS_{TE} {the total effective range shell polygon (2× coverage)}

3.2.4 Regions of Triple Coverage

The method for determining the total region of triple coverage is developed by expanding the double coverage method, as discussed in Section 3.2.3. An additional nested loop allows for the computation of the region of triple coverage between the sensor regions denoted in each of the $q_3(n)$ index triplets.

Algorithm 3.6 outlines the procedure of total triple coverage region determination. Note the addition of a conditional break statement depending upon whether intermediate region $RS_{i_1 i_2}$ is found to be empty (line 10). The additional check reduces unnecessary clip operations between non-comparable regions (this is discussed in detail at the end of Section 2.3.6).

Algorithm 3.6 *TripleCoverage*($\mathbf{RS}_E$)

1: n = length of $\mathbf{RS}_E$
2: **if** $n < 3$ **then**
3: error, necessary condition: $n \geq 3$
4: **end if**
5: initialize RS_{TE} {an empty polygon structure}
6: **for** $i_1 = 1$ to $n - 2$ **do**
7: **for** $i_2 = i_1 + 1$ to $n - 1$ **do**
8: **for** $i_3 = i_2 + 1$ to n **do**
9: $RS_{E_{i_1 i_2}} = \mathbf{RS}_E(i_1) \cap \mathbf{RS}_E(i_2)$
10: **if** $RS_{E_{i_1 i_2}}$ is empty **then**
11: break {triple coverage is impossible between (i_1, i_2, i_3)}
12: **end if**
13: $RS_{E_{i_1 i_2 i_3}} = RS_{E_{i_1 i_2}} \cap \mathbf{RS}_E(i_3)$
14: **if** RS_{TE} is empty **then**
15: $RS_{TE} = RS_{E_{i_1 i_2 i_3}}$
16: **else if** $RS_{E_{i_1 i_2 i_3}}$ is empty **then**
17: do nothing
18: **else**
19: $RS_{TE_{\text{temp}}} = RS_{TE} \cup RS_{E_{i_1 i_2 i_3}}$
20: $RS_{TE} = RS_{TE_{\text{temp}}}$
21: **end if**
22: **end for**
23: **end for**
24: **end for**
25: **return** RS_{TE} {the total effective range shell polygon ($3\times$ coverage)}

3.2.5 Regions of Arbitrary Coverage Multiplicity

When generalizing the methods of determination of total coverage regions, the approach used in the single, double, and triple coverage methods is not directly applicable in an elegant fashion. Experience gained from development of the aforementioned methods leads to a slightly different approach toward implementation for finding regions of an arbitrary coverage multiplicity.

Although the algorithm discussed in this section has the advantage of generality, it is slightly slower in execution than the fixed coverage multiplicity algorithms (single, double, and triple) described in Sections 3.2.2 through 3.2.4. Results in Section 3.4.1 show that there is typically a 1-2% performance penalty for using the arbitrary coverage multiplicity coverage model. Thus, in analyses specifically interested in single, double, or triple coverage, the aforementioned fixed coverage multiplicity algorithms are favorable.

3.2.5.1 Recursion vs. Iteration

The preceding algorithms for the determination of single, double, and triple coverage regions feature p nested loops, that iterate upon p integer indices. This is simple to implement for a known and fixed value of p. Recursion is the most obvious way (to the investigator) to achieve similar behavior for a p value that is not necessarily known at runtime. However, for larger values of p, this could add overhead, and increase runtime. Thus, it is desired to develop a purely iterative method instead. Rather than varying the indices as loop variables, p-dimensional vectors of integers, *index* and *ubound* are defined as

$$index = \begin{pmatrix} 1 & 2 & 3 & \dots & p \end{pmatrix} \text{(initial value only)}, \tag{3.1}$$

$$ubound = \begin{pmatrix} n-p+1 & n-p+2 & n-p+3 & \dots & n \end{pmatrix} \text{(fixed value)}. \tag{3.2}$$

The vector *index* is updated at every major iteration to reflect the indices of the set of p satellites being analyzed for $p\times$ coverage. The vector *ubound* reflects the upper bound on each digit in *index* (these values are fixed for a given p and n).

3.2.5.2 Index Generation

Initially, a very literal interpretation of the indices on the finite union operators, shown in Equation 2.17, was used to generate a comprehensive list of p-tuple set indices. This list contained $q_p(n)$ sets of p-dimensional vectors of integers. This list of vectors was generated using an intricate structure of loops, saving the values of each resulting set of indices as it executed. This implementation led to a very inefficient code, and it was found that the majority of computer runtime for certain problems was spent simply generating the indices. Additionally, the amount of memory required to store this data was considerable.

Consider the case of a constellation with $n = 25$ satellites being analyzed for $p = 8$ coverage. Using the expressions from Section 2.3.5, $q_8(25) = 1,081,575$ unique octuple sets, with $p = 8$ integers in each set. This results in a total of $8,652,600$ integers for storage. Assuming 16-bit (short) integers are used (this is more than adequate because each integer lies between 1 and $n = 25$), the minimum amount of memory required is 17.3 MB. Allocation, deallocation, and repeated access of this much memory are unrealistic burdens for a task that may need to be performed hundreds or thousands of times per second as n or p vary. At a minimum, the variations may occur at every function call in certain optimization problems.

Instead of this comprehensive *index* set, describing every *index* that may possibly be used, it was decided to transition to an ephemeral set, that consists of a single vector that is updated iteratively. The *index* memory storage for the case of $n = 25$ and $p = 8$ is thereby reduced from 17.3 MB to a mere 16 bytes – the space required to store 8 short integers. Thus, in general, given p 16-bit integers, at

1 byte per 8 bits, only $2p$ bytes are required for index storage.

The algorithm used to accomplish this is shown in Algorithm 3.7. The rightmost integer that has not reached its upper bound is found and incremented by one. If it was *not* the rightmost digit, the digits to its right are reset to count forward from the incremented value. Once one digit has been incremented (and any neighbors to the right are reset), the current *index* update is complete, and the value is returned.

For example, consider the case of a constellation with $n = 5$ satellites being analyzed for $p = 3$ coverage. The vector *index* takes on an initial value of (1 2 3), while *ubound* takes on a fixed value of (3 4 5) (as defined in Equations 3.1 and 3.2). Table 3.3 shows all values *index* takes on, along with comments on each increment.

As mentioned at the beginning of Section 3.2.5, performance of the arbitrary multiplicity algorithm is slightly inferior to the performance of the fixed multiplicity

Algorithm 3.7 *IncrementIndex*$(index, ubound)$

```
1:  p = length of index
2:  for j = p, p − 1, p − 2, . . . , 1 do
3:     if index(j) < ubound(j) then
4:        index(j) = index(j) + 1
5:        if j < p then
6:           {reset digits to the right of the j-th digit}
7:           for k = j + 1 to p do
8:              index(k) = index(k − 1) + 1
9:           end for
10:       end if
11:       break
12:    end if
13: end for
14: return index
```

Table 3.3: Index Values for $n = 5$, $p = 3$

i_1	i_2	i_3	Comment
3	4	5	*ubound*
1	2	3	increment i_3
1	2	4	increment i_3
1	2	5	i_3 at upper bound, increment i_2, reset i_3
1	3	4	increment i_3
1	3	5	i_3 at upper bound, increment i_2, reset i_3
1	4	5	i_2, i_3 at upper bound, increment i_1, reset i_2, i_3
2	3	4	increment i_3
2	3	5	i_3 at upper bound, increment i_2, reset i_3
2	4	5	i_2, i_3 at upper bound, increment i_1, reset i_2, i_3
3	4	5	i_1, i_2, i_3 at upper bound – final index value

algorithms. One of the primary reasons for this is the additional step of incrementing the index, as opposed to simply iterating upon loop variables. This performance decrease is not substantial (typically 1-2%), but varies slightly depending on the number of satellites involved.

3.2.5.3 Total Effective Range Shell, RS_{TE}

Using this new approach, avoiding excessive nested loops, the determination of regions of arbitrary coverage multiplicity becomes straightforward. Because the number of different values *index* may take on is unknown at runtime (for large p, in particular), a 'while' loop is used to iterate through values of *index*. The final value of *index* is arrived upon when the first element is at its upper bound, *ubound*(1). For each value of *index*, the algorithm iterates to perform the $p - 1$ intersection operations to find a p-multiplicity region of coverage between the sensor regions enumerated in the current value of *index*. Subsequently, each p-multiplicity region

of coverage from each *index* value is joined by union to the polygon structure RS_{TE}. Once all valid values of *index* have been processed, RS_{TE} represents the total region of $p\times$ coverage between the satellites defined in the $\mathbf{RS}_E$ array of effective range shells. Algorithm 3.8 shows how this process is implemented.

3.2.6 Clipping with Altitude Shell and Computing Area

Once a total effective range shell polygon, RS_{TE}, has been produced, the final operations can be performed to find the actual area of coverage at the desired coverage multiplicity within the area of interest. First, a polygon in the shape of an annulus is defined with inner and outer radii corresponding to the lower (LTAS) and upper target altitude shell (UTAS) radii, r_l and r_u respectively. This process is straightforward, and is discussed and illustrated briefly in Section 2.1.3.3. The resulting procedure is outlined in Algorithm 3.9.

Once AS is produced, a single intersection operation is performed with RS_{TE}. The area of the resulting region, RS_{AS} can be computed readily using the method given at the end of Section 2.3.1. Algorithm 3.10 illustrates this brief process.

3.2.7 Complete Numerical ATH Coverage Model

Combining the algorithms presented in Sections 3.2.1 through 3.2.6, the ATH coverage for a planar constellation at any desired coverage multiplicity can be evaluated. Consider a set of example parameters, and the associated procedure, shown in Algorithm 3.11 (which shows a typical process for the evaluation of ATH

Algorithm 3.8 *ArbitraryCoverage*($\mathbf{RS}_E, p$)

1: n = length of $\mathbf{RS}_E$
2: **if** $n < p$ **then**
3: error, necessary condition: $n \geq p$
4: **end if**
5: initialize RS_{TE} {an empty polygon structure}
6: initialize $index, ubound$ {as in Equations 3.1 and 3.2}
7: initialize $SetsRemaining = 1, MakeNextIndex = 1$
8: **while** $SetsRemaining == 1$ **do**
9: **if** $index(1) == ubound(1)$ **then**
10: $SetsRemaining = 0, MakeNextIndex = 0$
11: **end if**
12: initialize $T_1 = \mathbf{RS}_E(index(1))$ {first sensor region enumerated in $index$}
13: initialize T_2 {an empty polygon structure}
14: **for** $j = 2$ to p **do**
15: **if** T_1 is empty **then**
16: break {p-multiplicity coverage impossible with $index$}
17: **else**
18: $T_2 = T_1 \cap \mathbf{RS}_E(index(j))$
19: $T_1 = T_2$
20: **end if**
21: **end for**
22: **if** T_1 is NOT empty **then**
23: **if** RS_{TE} is NOT empty **then**
24: $RS_{TE_{temp}} = RS_{TE} \cup T_1$
25: $RS_{TE} = RS_{TE_{temp}}$
26: **else**
27: $RS_{TE} = T_1$
28: **end if**
29: **end if**
30: **if** $MakeNextIndex == 1$ **then**
31: $index$ = *IncrementIndex*($index, ubound$) {get next index vector}
32: **end if**
33: **end while**
34: **return** RS_{TE} {the total effective range shell polygon ($p\times$ coverage)}

Algorithm 3.9 *GenerateAltitudeShell*(r_l, r_u, m)

initialize AS {an empty polygon structure}
for $i = 1$ to m **do**
  $\theta_i = \frac{(i-1)(2\pi)}{m}$
  $(x_{\text{inner}_i}, y_{\text{inner}_i}) = (r_l \cos\theta_i, r_l \sin\theta_i)$
  $(x_{\text{outer}_i}, y_{\text{outer}_i}) = (r_u \cos\theta_i, r_u \sin\theta_i)$
  add $(x_{\text{inner}_i}, y_{\text{inner}_i})$ to AS as the i-th vertex of the inner contour
  add $(x_{\text{outer}_i}, y_{\text{outer}_i})$ to AS as the i-th vertex of the outer contour
end for
return AS {dual-band altitude shell polygon structure}

Algorithm 3.10 *EvaluateCoverageInAS*(r_l, r_u, RS_{TE}, m)

$AS =$ *GenerateAltitudeShell*(r_l, r_u, m)
$C_{p\times} = RS_{TE} \cap AS$ {region of $p\times$ coverage in AS}
return area inside $C_{p\times}$ {see Section 2.3.1 and Equation 2.14}

coverage). The example shown illustrates analysis for single, double, and triple coverage between satellites with omni-directional sensors equally distributed in a single circular orbit.

The resulting coverage areas are tabulated in Table 3.4, and the corresponding configuration is illustrated in Figure 3.12. The computed area denoted as 'single

Algorithm 3.11 Example Problem – Circular Orbit, Equal Spacing in Longitude

$r_t = 6500$ km, $r_l = 7400$ km, $r_u = 11400$ km, $R = 5000$ km, $r_s = 7000$ km
$n = 10$ {number of satellites}
$m = 100$ {polygon resolution, PPC}
$p_{\min} = 1$ {lower multiplicity of interest}
$p_{\max} = 3$ {upper multiplicity of interest}
for $p = p_{\min}$ to $p_{\max}$ **do**
  $\mathbf{RS}_E =$ *GenerateRangeShellArray1*(r_t, r_s, R, n, m) {eff. range shell array}
  $RS_{TE} =$ *ArbitraryCoverage*$(\mathbf{RS}_E, p)$ {determine total region of $p\times$ coverage}
  $A_p =$ *EvaluateCoverageInAS*(r_l, r_u, RS_{TE}, m)
end for

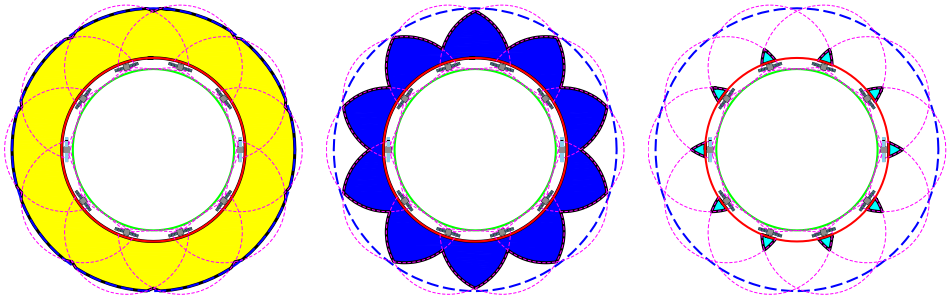
(at least)' in Table 3.4 refers to the area that has *at least* single coverage. This region is shown in Figure 3.12a. Observe that within the shaded single coverage region, there are smaller regions where multiple effective range shells (the boundaries centered on each satellite) overlap. Similarly, the area denoted by 'double (at least)' in Table 3.4 corresponds to the region of *at least* double coverage, shown in Figure 3.12b. Finally, the region of triple coverage (and *only* triple coverage – the highest coverage multiplicity in this example) is shown in Figure 3.12c. Combining these results, Figure 3.12d shows the highest coverage multiplicities present at all locations for the chosen parameters (these parameters were arbitrarily chosen and do not produce an optimal configuration in any sense).

Table 3.4: Computed Areas, Example in Algorithm 3.11

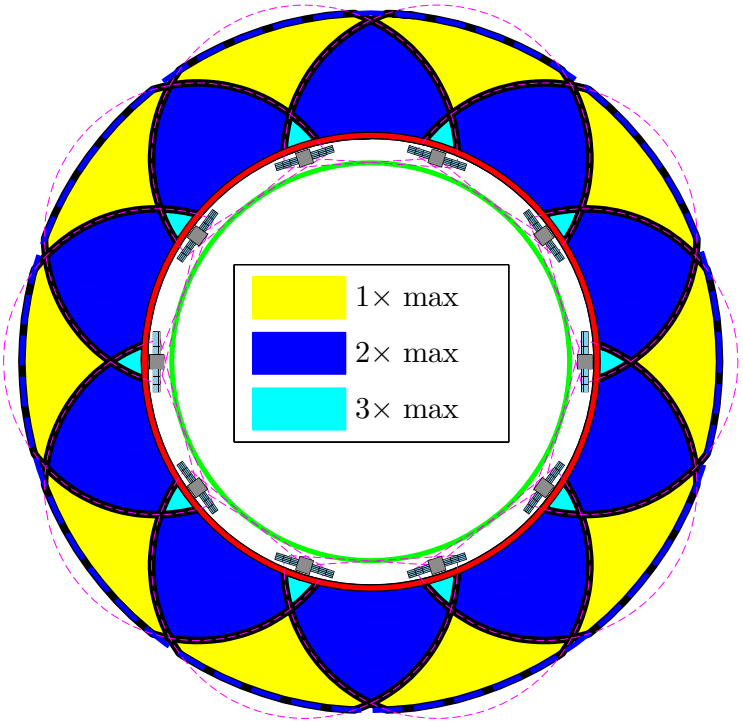
Coverage Multiplicity, p	Area (km^2)
Single (at least)	234.90×10^6
Double (at least)	144.72×10^6
Triple	7.0810×10^6

3.3 Validation and Error Analysis

All clipping implementations used for this research (with the conditional exception of Jacquenot's code)[3] provide the same answer in terms of determining regions of polygon overlap and computing their areas. The only variations that occur are on the order of round-off and truncation error, and are a result of the different ways that certain intermediate calculations are performed. Thus, the primary source of error that is cause for concern is the error introduced when discretizing the various curvilinear boundaries of regions of interest into finite sets of polygon vertices. This



(a) Single (at least) (b) Double (at least) (c) Triple



(d) Highest Coverage Multiplicities at All Locations in AS for Chosen h_s

Figure 3.12: Satellite Configuration, Example in Algorithm 3.11

topic is discussed for the general case of a circle approximated by an inscribed polygon in Section 2.1.2.1. As is shown in this section, the results presented there provide an acceptable estimate of relative error, but will differ slightly because some curvilinear boundaries are concave, rather than purely convex (as is the case for a circle).

The aforementioned conditional exception in the case of Jacquenot's *Polygon Intersection* code[3] occurs only if a non-zero tolerance parameter is specified. The reason for this is discussed thoroughly in Section 3.3.1.

3.3.1 *Polygon Intersection* Tolerance Parameter

Jacquenot's *Polygon Intersection*[3] code does not only process the polygon regions before and after interfacing with GPC[6] via MEX, it also incorporates an additional tolerance parameter (let it be denoted by ϵ). The implementation of this tolerance parameter is (as is often the case) poorly documented. Consider a general case where there are l polygon regions described in the input data structure. From inspection of the code, it is determined that the tolerance parameter is implemented by neglecting regions that are smaller than $\epsilon \max A_i$ for all $i \in \{1, 2, 3, \ldots, l\}$, i.e. ϵ multiplied by the area of the largest input polygon region. Two tolerance values are investigated – 1×10^{-6} (default value) and 0. As provided by Jacquenot, the code does not allow for 0 tolerance, and requires modification to allow for it. There is no discernible effect upon performance when using either value.

First, consider the case of a numerical analysis using a 100 PPC polygon resolution. The analytical and numerical results (at either tolerance of 1×10^{-6}

or 0) both produce a cosmetically identical area of coverage vs. altitude curve, as shown in Figure 3.13. Similar area of coverage vs. altitude curves for 1000 and 4000 PPC analyses (unsurprisingly) also appear cosmetically identical, and are consequently not reproduced here. Thus, it is more illustrative to discuss the relative error between the two curves.

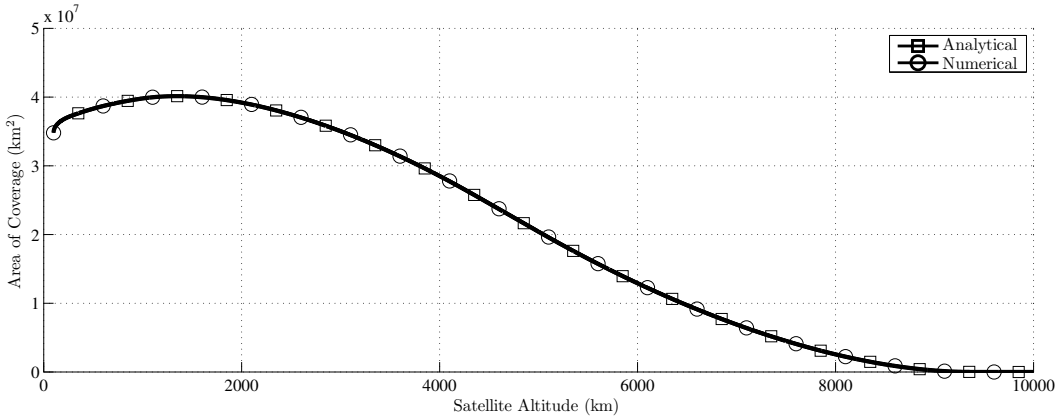


Figure 3.13: Comparison of Analytical and Numerical Coverage Area Curves – 100 PPC

Figure 3.14 shows the percent relative error (relative to the analytically obtained values) in area computation for the case of 100 PPC with a tolerance of 1×10^{-6}.

Despite being cosmetically identical in Figure 3.13, Figure 3.14 shows relative error very slowly increasing as the coverage area vanishes and 'blowing-up' abruptly to 100%. This maximum in relative error corresponds to a condition where the numerically computed area is zero, while the analytical area is non-zero, i.e. by the

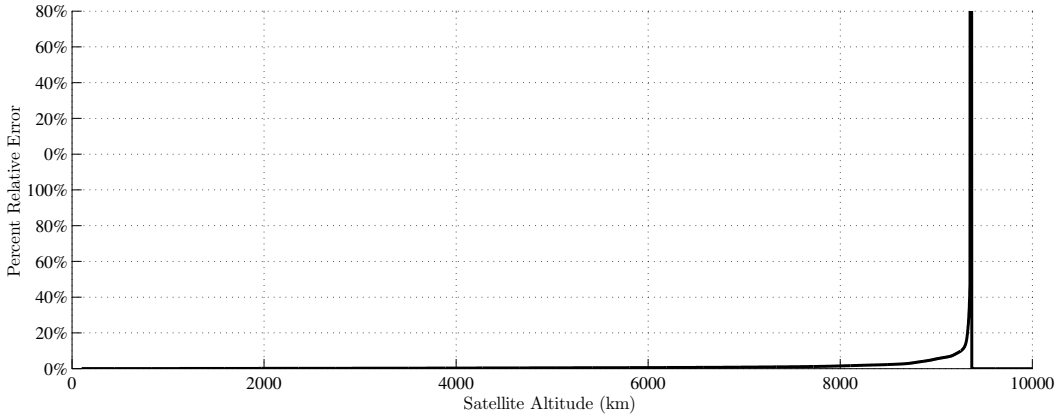


Figure 3.14: Percent Error Between Analytical and Numerical – 100 PPC, 1×10^{-6} Tolerance

relation

$$A_{\text{err}} = \frac{|A_{\text{A}} - A_{\text{N}}|}{A_{\text{A}}}. \tag{3.3}$$

where A_{A}, A_{N}, and A_{err} represent the analytically computed coverage area, numerically computed coverage area, and the relative error in coverage area computation respectively.

The reverse case, where the numerically computed area is found to be non-zero with an analytical area of zero, is not encountered in this investigation. This is primarily due to the use of polygons inscribed inside convex curvilinear regions (the majority of the coverage region boundaries turn out to be convex), rather than polygons encompassing them – thus for any region along the convex boundary of a polygon, numerically computed enclosed area will be less than the analytical result.

The implementation of the tolerance parameter in Jacquenot's code[3] (briefly discussed at the beginning of this section) is partly to blame for the extent of the

regime over which this relative error of 100% can occur. Consider Tables 3.5 and 3.6 – using the default tolerance of 1×10^{-6}, the computed numerical area is determined to be zero by the $i = 9252$ data point (Table 3.5), whereas with a zero tolerance, the numerical area is only found to be zero by the $i = 9268$ data point (Table 3.6), and the reported numerical area is more accurate to the analytical area up until that point (rather than just being rounded to zero).

Table 3.5: Data Excerpt – 100 PPC, 1×10^{-6} Tolerance

i	h (km)	A_{A} (km^2)	A_{N} (km^2)	$\%A_{\mathrm{err}}$
⋮	⋮	⋮	⋮	⋮
9250	9349.9	690.3	536.0	22.35
9251	9350.9	623.1	476.9	23.46
9252	9351.9	559.3	0.0	100.00
9253	9352.9	499.0	0.0	100.00
9254	9353.9	442.0	0.0	100.00
9255	9354.9	388.5	0.0	100.00
9256	9355.9	338.5	0.0	100.00
9257	9356.9	291.9	0.0	100.00
9258	9357.9	248.7	0.0	100.00
9259	9358.9	208.9	0.0	100.00
9260	9359.9	172.7	0.0	100.00
9261	9360.9	139.8	0.0	100.00
9262	9361.9	110.4	0.0	100.00
9263	9362.9	84.5	0.0	100.00
9264	9363.9	62.0	0.0	100.00
9265	9364.9	43.0	0.0	100.00
9266	9365.9	27.5	0.0	100.00
9267	9366.9	15.4	0.0	100.00
9268	9367.9	6.8	0.0	100.00
9269	9368.9	1.7	0.0	100.00
9270	9369.9	0.0	0.0	0.00
9271	9370.9	0.0	0.0	0.00
⋮	⋮	⋮	⋮	⋮

Table 3.6: Data Excerpt – 100 PPC, 0 Tolerance

i	h (km)	A_{A} (km^2)	A_{N} (km^2)	$\%A_{\mathrm{err}}$
⋮	⋮	⋮	⋮	⋮
9250	9349.9	690.3	536.0	22.35
9251	9350.9	623.1	476.9	23.46
9252	9351.9	559.3	421.2	24.68
9253	9352.9	499.0	369.0	26.04
9254	9353.9	442.0	320.3	27.54
9255	9354.9	388.5	275.0	29.22
9256	9355.9	338.5	233.1	31.12
9257	9356.9	291.9	194.7	33.28
9258	9357.9	248.7	159.8	35.75
9259	9358.9	208.9	128.3	38.60
9260	9359.9	172.7	100.2	41.94
9261	9360.9	139.8	75.7	45.89
9262	9361.9	110.4	54.5	50.64
9263	9362.9	84.5	36.8	56.42
9264	9363.9	62.0	22.6	63.56
9265	9364.9	43.0	11.8	72.50
9266	9365.9	27.5	4.5	83.56
9267	9366.9	15.4	0.7	95.73
9268	9367.9	6.8	0.0	100.00
9269	9368.9	1.7	0.0	100.00
9270	9369.9	0.0	0.0	0.00
9271	9370.9	0.0	0.0	0.00
⋮	⋮	⋮	⋮	⋮

Clearly, the implementation of the tolerance parameter is such that a non-zero tolerance is detrimental to accuracy within the context of this problem. Thus, when creating all the remaining data presented in this section, a tolerance of 0 is used. However, it is clear that even with a zero tolerance (as expected), a significant amount of relative error can develop as the coverage region vanishes. The absolute error is still quite small, as can be seen in the data excerpt shown in Table 3.6, but

it is still instructive to characterize the source.

Consider data point $i = 9265$ shown in Table 3.6 where there is a 72.5% discrepancy between the analytical and numerical coverage areas. Figure 3.15 shows the associated coverage regions at 100 and 4000 PPC polygon resolutions. Note that the inset diagrams are shown at the same scale. Observe that in the 4000 PPC case (with 40× as many vertices) the UTAS follows a much more gradual curve, and thus, the convex boundary of the region of interest region extends further into the effective range shell, resulting in a larger coverage area.

Although the 4000 PPC case is not exactly representative of the analytical case, it is a substantial improvement in accuracy, and serves to illustrate the cause of the increase in relative error as the region of coverage vanishes. Figure 2.3 shows that for 100 PPC, relative error compared to the analytical case is expected to be substantially less than 0.1%, while the 4000 PPC case should be well within 0.0001% relative error.

This demonstrates (and agrees with intuition, and previous discussion in Section 2.1.2.1) that higher polygon resolutions provide greater accuracy. Percent relative error plots are shown for 1000 and 4000 PPC in Figures 3.16 and 3.17 respectively. It is, however, deceptive that these two plots do not show any data points at which relative error becomes 100% as the coverage area vanishes – in fact, if small enough steps in increasing altitude are taken for *any* finite polygon resolution, an altitude can (with few exceptions) be found at which the percent relative error is 100%. It is only by coincidence that this regime falls between data points in altitude on Figures 3.16 and 3.17.

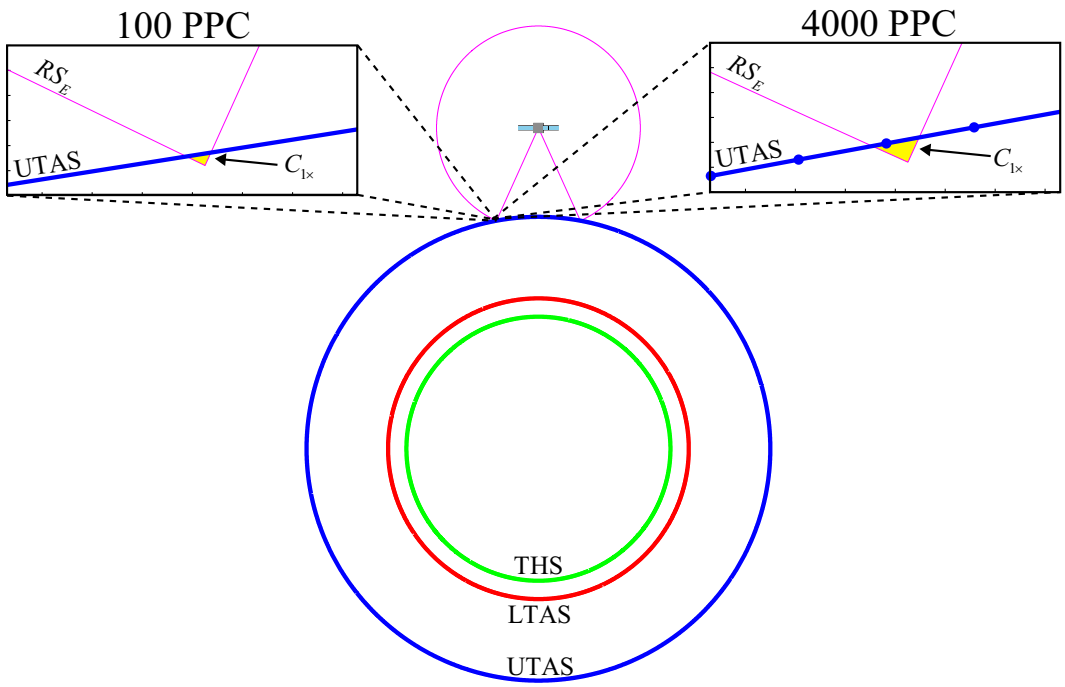


Figure 3.15: Coverage Configuration at $i = 9265$ (see Table 3.6)

One case that could be an exception is the extremely rare condition that the x-coordinate of the lower corner of the range shell is perfectly matched (to within machine ε) to the x-coordinate of an upper target altitude shell (UTAS) vertex. Thus, the region vanishes simultaneously with the analytical case as altitude increases. The computed areas under such circumstances will still be different, as the numerical case presents a vanishing quadrilateral coverage region, as opposed to one with curvilinear boundaries as in the analytical case.

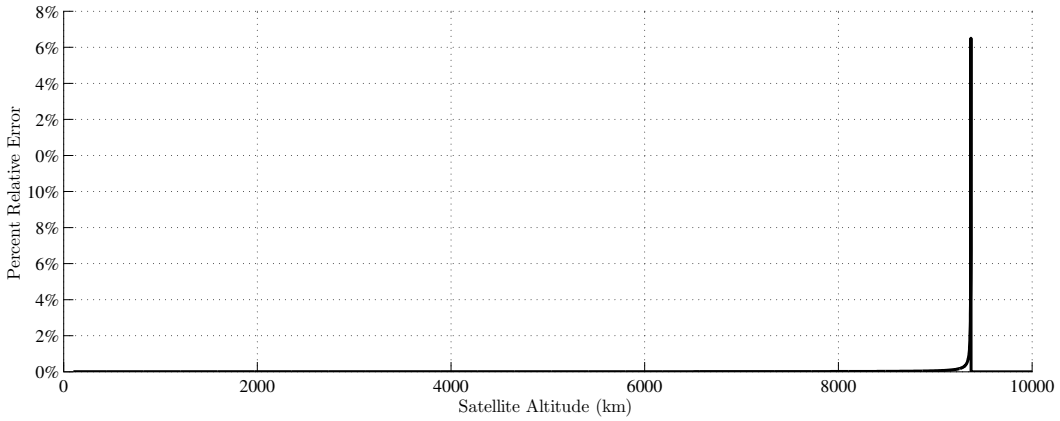


Figure 3.16: Percent Error Between Analytical and Numerical – 1000 PPC, 0 Tolerance

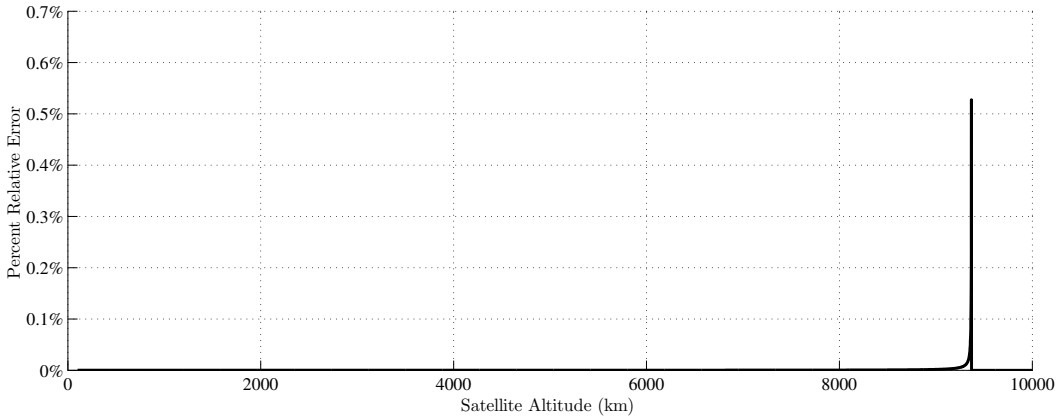


Figure 3.17: Percent Error Between Analytical and Numerical – 4000 PPC, 0 Tolerance

3.3.2 Marchand and Kobel's 'Example 1' – Non-Vanishing Regions

The majority of error analysis in this research is carried out for the single satellite case, built upon Marchand and Kobel's[4] 'Example 1.' Because this is (at the time of this writing) the only model constructed to analytically compute in-plane

coverage areas, it is the favored metric for comparison with the numerical ATH coverage models (re-derivation of this analytical model is presented in Appendix A).

3.3.2.1 Error in Area Calculation

The case discussed in Section 3.3.1 analyzes relative error over the entire range of altitude investigated by Marchand and Kobel's[4] 'Example 1.' That discussion is primarily centered upon the 'blow-up' of relative error as the coverage region vanishes. However, from the values listed in Table 3.6, it is clear that, while the relative error becomes large, the absolute error in this region is very small compared the total area in the vicinity of maximum coverage. Thus, it is useful to analyze the relative error solely in the vicinity of maximum coverage, and to determine how the actual amount of error relates to the estimation of error discussed Section 2.1.2.1.

Figures 3.18 through 3.21 show percent relative error in in-plane coverage area calculation vs. satellite altitude from $h_s = h_t$ to 5000 km for 10, 100, 1000, and 4000 PPC respectively.

According to the estimate made in Section 2.1.2.1, a polygon resolution of 100 PPC is expected to exhibit a relative error on the order of 0.1%. Figure 3.19 shows that this estimate is reasonable, particularly in the vicinity of the altitude of maximum coverage, at $h_s = 1349$ km. However, as satellite altitude increases, a monotonic rise in relative error is observed regardless of polygon resolution. The reason for this will become clear in the ensuing discussion.

For polygon resolutions of 100, 1000, and 4000 PPC, shown in Figures 3.19,

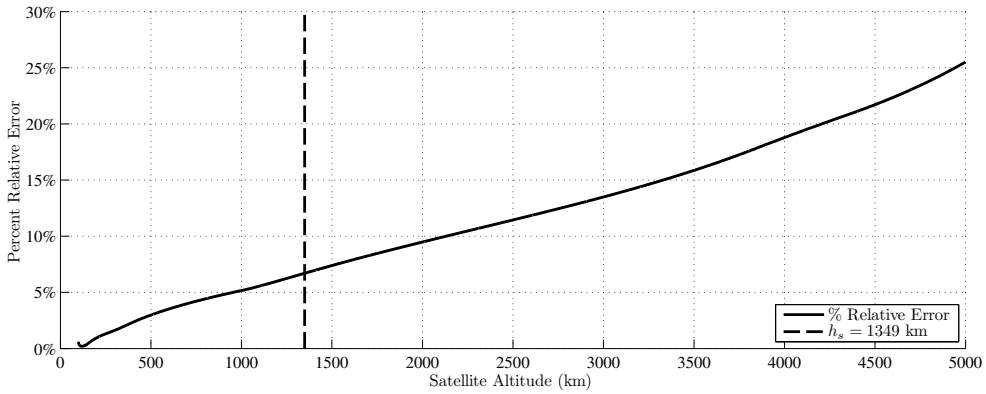


Figure 3.18: Relative Error vs. Satellite Altitude for $h_s = h_t$ to 5000 km – 10 PPC

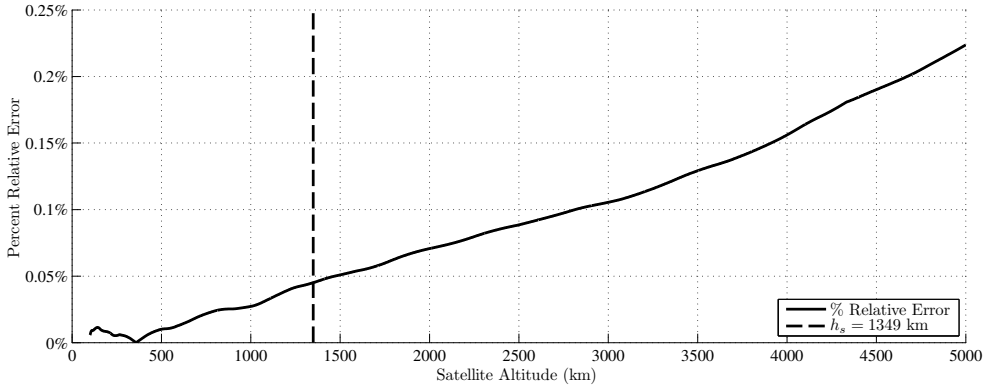


Figure 3.19: Relative Error vs. Satellite Altitude for $h_s = h_t$ to 5000 km – 100 PPC

3.20, and 3.21, respectively, a noticeable local minimum in percent relative error exists at approximately $h_s = 365$ km. Plotting this configuration, as shown in Figure 3.22, reveals that this minimum in relative error results from an interaction between the convex and concave boundaries of the coverage region.

Figure 3.23 shows an exaggerated case using an exceptionally low polygon resolution (20 PPC). The upper boundaries of the coverage region are convex, and

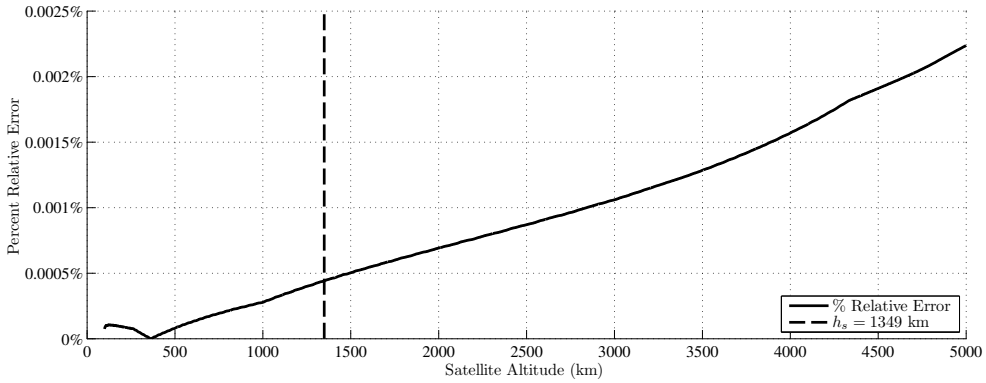


Figure 3.20: Relative Error vs. Satellite Altitude for $h_s = h_t$ to 5000 km – 1000 PPC

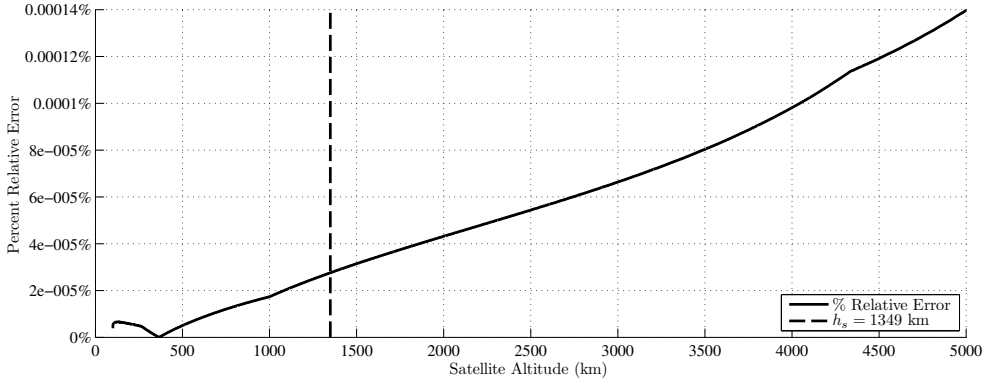


Figure 3.21: Relative Error vs. Satellite Altitude for $h_s = h_t$ to 5000 km – 4000 PPC

consequently, the polygon representation along those boundaries is inscribed inside the true 'analytical' curves. As a result, the polygon underestimates the extent of the true region in that vicinity. In contrast, the lower boundary is concave, and the polygon edges circumscribes the true boundary, overestimating the extent of the true region. It is an interesting coincidence that this particular satellite altitude

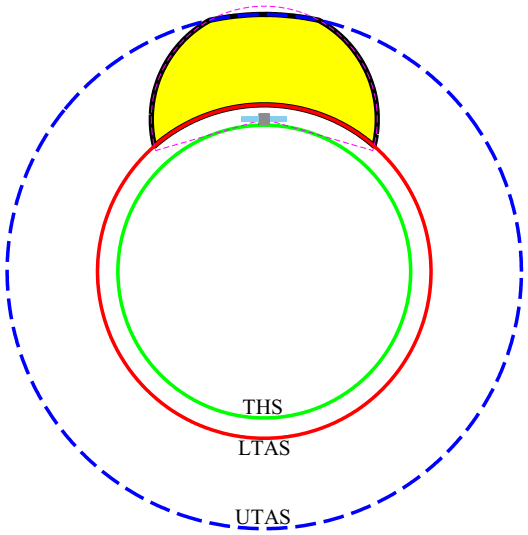


Figure 3.22: Configuration at $h_s = 365$ km

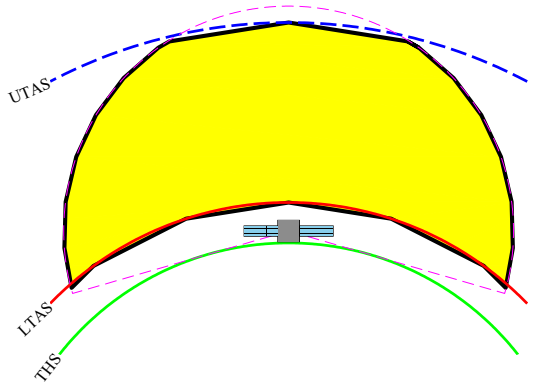


Figure 3.23: Configuration at $h_s = 365$ km – 20 PPC, Detail

(365 km) corresponds to a balance between the overestimation and underestimation of areas along the concave and convex boundaries respectively.

This behavior is one of the major factors in the monotonic rise in error after this point in Figures 3.18 through 3.21. Consider a higher altitude (but still within

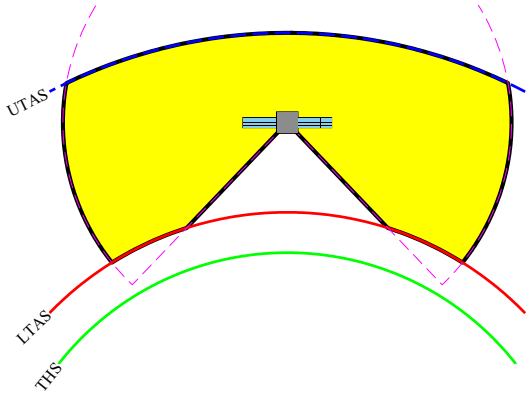


Figure 3.24: Configuration at $h_s = 3000$ km

the interval from h_t to 5000 km) – i.e. $h_s = 3000$ km. Figure 3.24 illustrates the configuration. Figure 3.19 shows that $h_s = 3000$ km lies in the vicinity where relative error increases proportional to satellite altitude.

As satellite altitude increases, the length of the coverage boundary on the lower target altitude shell (LTAS) decreases, limiting the extent of the only concave boundary on the sensor region. Additionally, the presence of the missing 'slice' due to the tangent height triangle (THT) removes the bulk of the remaining concave boundary. This 'slice' grows starting at $h_s = h_l = 1000$ km, and causes a slight corner in the curve in Figures 3.20 and 3.21. Thus, the convex boundaries dominate as a source of error, but in one 'direction' only, causing an overall underestimation of the true coverage region.

Overall, and especially in the vicinity of maximum coverage, relative error in area approximation behaves as estimated in Section 2.1.2.1.

3.3.2.2 Error in Altitude of Maximum Coverage

Marchand and Kobel's[4] 'Example 1' does not *only* use their result to compute coverage areas at various satellite altitudes, but uses it as an objective function for determining the satellite altitude providing maximum in-plane coverage. Using the analytical implementation described in Appendix A, the maximum in-plane coverage area is found to be approximately $A = 4.0146 \times 10^7$ km^2, occurring at a satellite altitude of $h_s = 1348.876$ km (determined to a resolution in altitude of 0.3 m).

The problem is re-solved using the numerical algorithms described in Section 3.2, using polygon resolutions varying from 10 to 4000 PPC. At each contour resolution, the grid is refined to a resolution of approximately 0.6 m in satellite altitude. The resulting altitudes of maximum coverage are summarized in Figure 3.25. The corresponding maximum areas of coverage for each polygon resolution are shown in Figure 3.26.

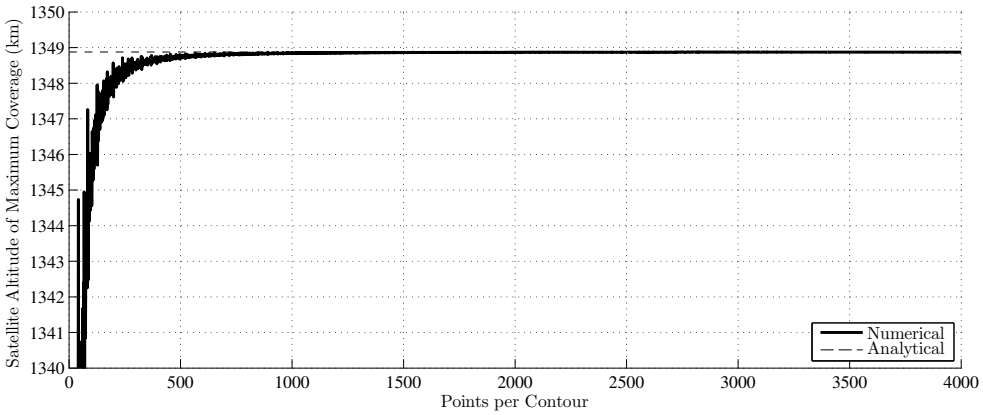


Figure 3.25: Satellite Altitude of Maximum Coverage

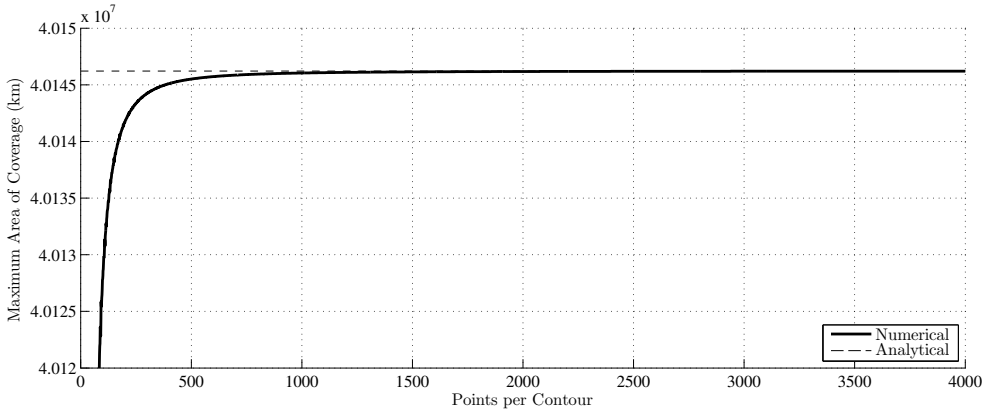


Figure 3.26: Maximum Area of Coverage

As with other discussions in this research regarding error, absolute error at higher polygon resolutions is quite small, so analyzing the percent relative error instead is far more useful. Figures 3.27 and 3.28 show the percent relative error in satellite altitude of maximum coverage, and the actual computed maximum coverage area.

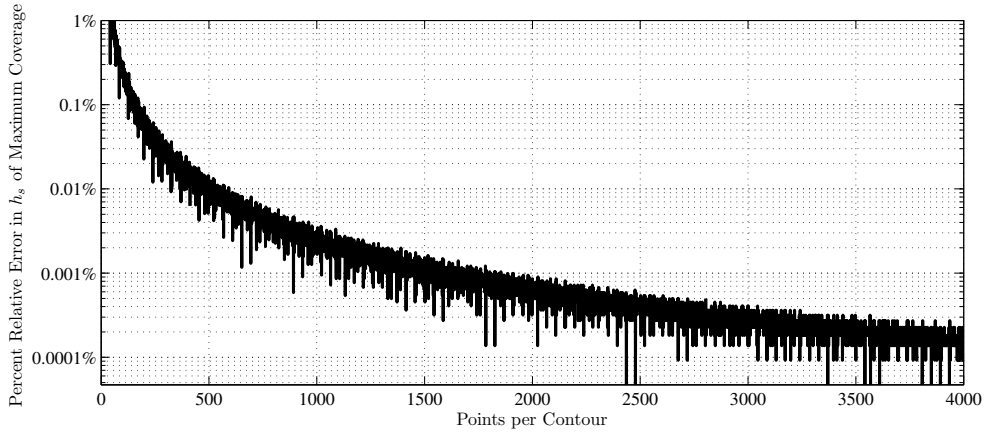


Figure 3.27: Relative Error in Satellite Altitude of Maximum Coverage

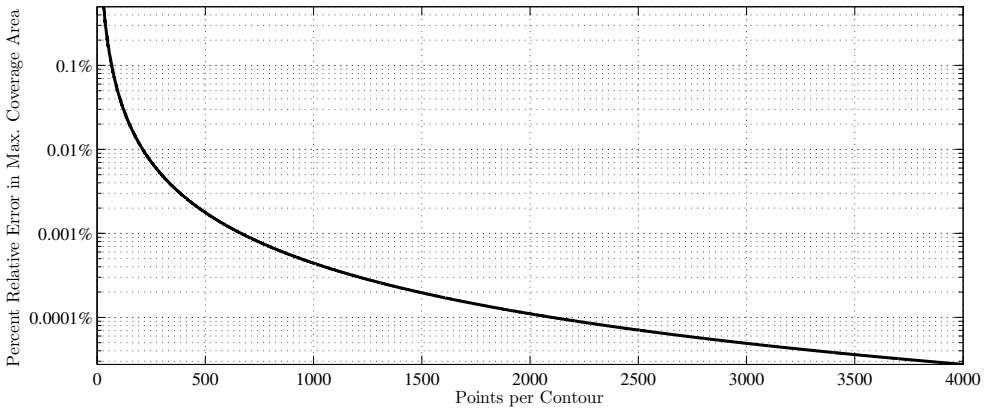


Figure 3.28: Relative Error in Maximum Area of Coverage

3.3.3 Marchand and Kobel's 'Example 1' – n-satellites

Generality is one of the primary advantages of numerical analysis of the ATH coverage problem. No special analytical cases must be rederived when the shapes and problem parameters change. As long as the desired in-plane coverage regions can be described and positioned as polygons, amounts of coverage at any coverage multiplicity can be determined using the numerical algorithms presented in Section 3.2. The case discussed in Section 3.3.2 mimics the analytical results developed by Marchand and Kobel.[4] However, at the time of this writing, any expansion upon that problem lacks a comparable analytical model for use in validation.

Fortunately, because the problem itself is easy to visualize, and because the area calculations are performed directly upon polygons that are available for visual inspection, it is straightforward to at least verify results are not grossly incorrect. For example, when analyzing for regions of higher coverage, the regions for which area is being evaluated can be easily illustrated, and the number of satellites and their

overlapping range shells can be manually counted to ensure the resulting regions correspond to the desired coverage multiplicity.

The most intuitive expansion to Marchand and Kobel's work[4] is to analyze the coverage due to n satellites, with omni-directional sensors, equally spaced in a circular orbit. This problem is time-invariant, just as in Marchand and Kobel's work, and its implementation is straightforward. An example algorithm for this scenario is discussed in Section 3.2.7.

Lacking an analytical reference model for this problem, a very high resolution numerical model is used instead. Discussion in Section 3.3.2 reveals that for the single satellite case (using Marchand and Kobel's parameters),[4] relative error in the relevant range of satellite altitudes is on the order of 0.1% at 100 PPC, and on the order of 0.0001% at 4000 PPC. This three order of magnitude improvement in relative error involves a two order of magnitude decrease in performance as a result of the increase in polygon resolution (see Table 3.2).

Because 100 PPC is selected as an acceptable trade between accuracy and performance, better characterization of error at this polygon resolution is of interest. Consider a 12 satellite constellation using the parameters shown in Table 3.7, that, aside from 11 additional satellites, correspond exactly to the Marchand and Kobel[4] parameters in use for the discussion in Section 3.3.2.

Table 3.7: Parameters for Numerical Comparison of a 12 Satellite Constellation

Parameter Value	Description
$n = 12$	number of satellites
$p = 1, 2, 3$	coverage multiplicities of interest
$R_e = 6378.14$ km	assumed Earth radius
$R = 5000$ km	omni-directional sensor range
$h_t = 100$ km	tangent height
$h_l = 1000$ km	lower altitude bound
$h_u = 5000$ km	upper altitude bound
$h_{s_{\min}} = 100$ km	lower limit on satellite altitude
$h_{s_{\max}} = 10000$ km	upper limit on satellite altitude

Computing the area of coverage at 4000 PPC at $p = 1, 2, 3$ for satellite altitudes ranging from $h_{s_{\min}}$ to $h_{s_{\max}}$ (at a resolution of 5 km) yields the plot shown in Figure 3.29. The 'single (at least)' curve corresponds to the area that exhibits *at least* single coverage (thus including double and triple coverage regions also). Similarly, the 'double (at least)' curve shows the area with *at least* double coverage (which includes regions of triple coverage). The 'triple' curve corresponds only to regions of triple coverage – the chosen problem parameters do not result in quadruple or higher coverage. To better illustrate this, consider Figure 3.30, that shows the single, double, and triple coverage regions at $h_s = 1500$ km.

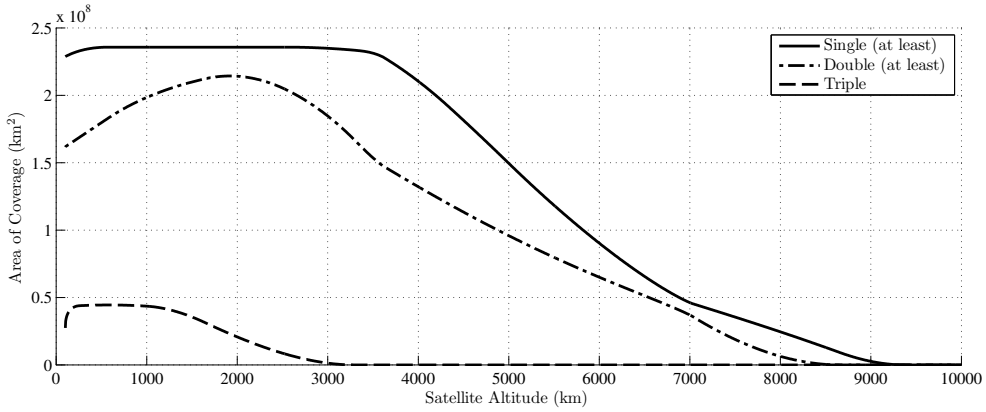
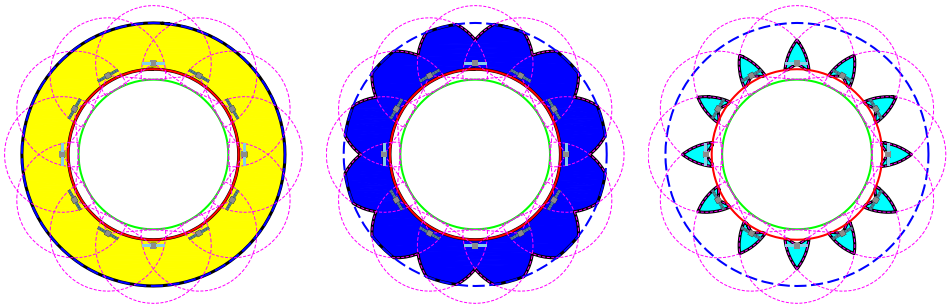
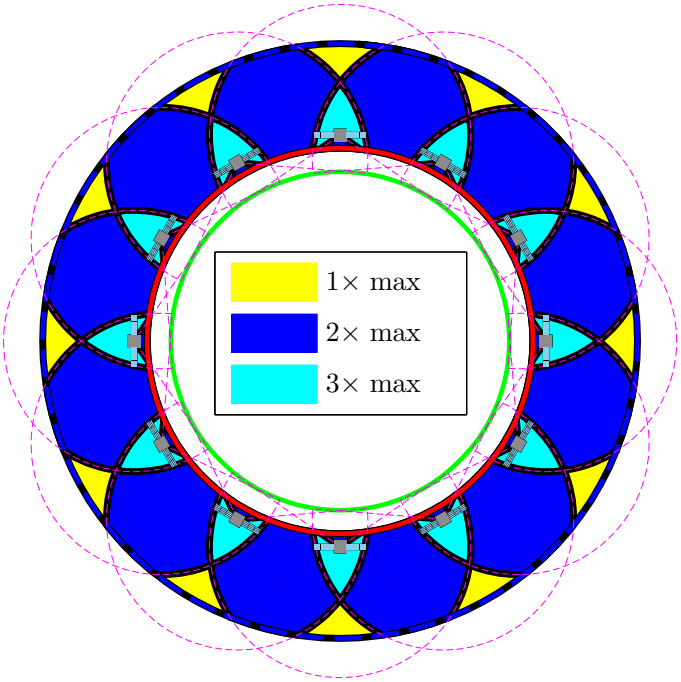


Figure 3.29: Single (at least), Double (at least), and Triple-Fold Areas of Coverage for the 12 Satellite Case Computed at 4000 PPC (see Table 3.7)



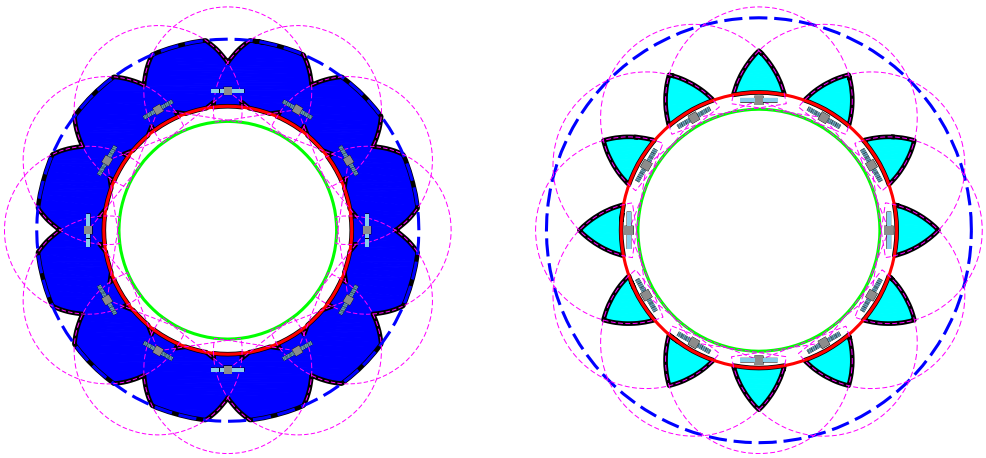
(a) Single (at least) (b) Double (at least) (c) Triple



(d) Highest Coverage Multiplicities at All Locations in AS for Chosen h_s

Figure 3.30: Coverage Regions for $h_s = 1500$ km

Figure 3.29 shows a plateau in single (at least) coverage area between the altitudes of approximately 580 and 2570 km. This corresponds to a total saturation of the dual-altitude band shell with single coverage. An example of this maximum single coverage case is shown in Figure 3.30a. A maximum in double (and above) coverage occurs at 1917 km, while a maximum in triple coverage occurs at 580 km. These cases are shown in Figures 3.31a and 3.31b respectively.



(a) Maximum in Double (at least) Coverage, $h_s = 1917$ km

(b) Maximum in Triple Coverage, $h_s = 580$ km

Figure 3.31: Configurations of Maximum Double (at least) and Triple Coverage

At 100 PPC, the curves representing coverage area are cosmetically identical to the curves describing the 4000 PPC case shown in Figure 3.29, and are thus not reproduced here. To differentiate between them, the relative error is analyzed instead. Recall that, as discussed in Section 3.3.1 (while discussing the impact of the tolerance parameter in Jacquenot's implementation)[3] as the coverage area vanishes, relative error 'blows-up.' The absolute error is small, and it occurs far

from the vicinity of maximum coverage, and thus this behavior is of passing interest. Figure 3.32 shows the relative error of the 100 PPC analysis compared to the 4000 PPC analysis (chosen as a reference model). Compare this plot with Figure 3.29: the sudden rises in relative error correspond to the vanishing of each coverage region multiplicity, as expected. Near their respective altitudes of maximum coverage, each multiplicity retains a well-bounded relative error, on the order of 0.1%, as expected.

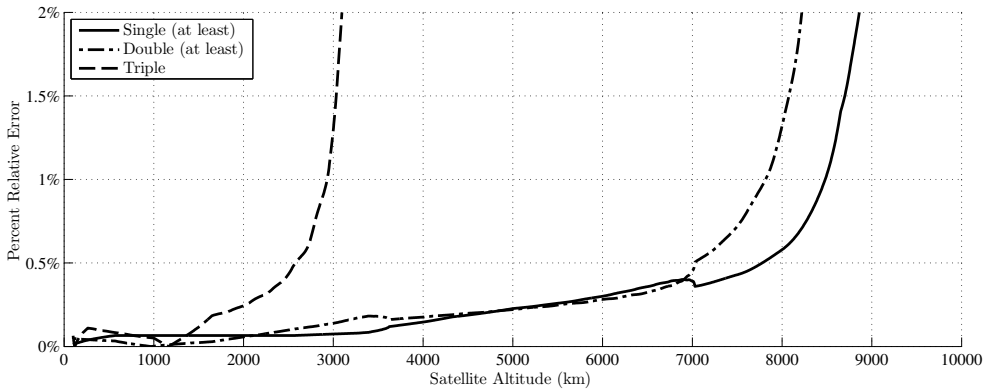


Figure 3.32: Relative Error at 100 PPC vs. 4000 PPC for Single, Double, and Triple Areas of Coverage

Although the 12-satellite example presented in this section demonstrates that 100 PPC is an acceptable polygon resolution to achieve the (admittedly arbitrary) desired relative error of 0.1% near the maximum coverage altitude, other problems may not be so well behaved. Each problem should be analyzed to determine if the chosen polygon resolution provides the desired accuracy. Unfortunately, this involves performing a time-consuming high-resolution analysis first. Depending upon the type of the analysis, this may prove to be a waste of time and effort. The example presented here is a simple analysis involving equal numbers of function eval-

uations at 100 and 4000 PPC. Clearly, the 4000 PPC analysis is more accurate, and if an accurate curve, such as in Figure 3.29 is the only desired result, re-computing the data at a less-accurate resolution is a waste of time. In such a case, the high resolution model would be used alone, assuming it can be computed in a reasonable amount of time. The motivation for analyzing the 12-satellite example in this section, however, is to demonstrate the amount of error present at 100 PPC (which is the only reason it is computed at all).

Performing a high-resolution analysis simply for error estimation would be beneficial in a problem where the low-resolution model is to be used much more frequently. An example of this would be a problem solved using a parameter optimization code requiring numerical derivatives – this greatly increases the necessary number of function evaluations, and it would then be advantageous if a lower polygon resolution can still achieve acceptable accuracy.

3.4 Numerical ATH Coverage Model Performance

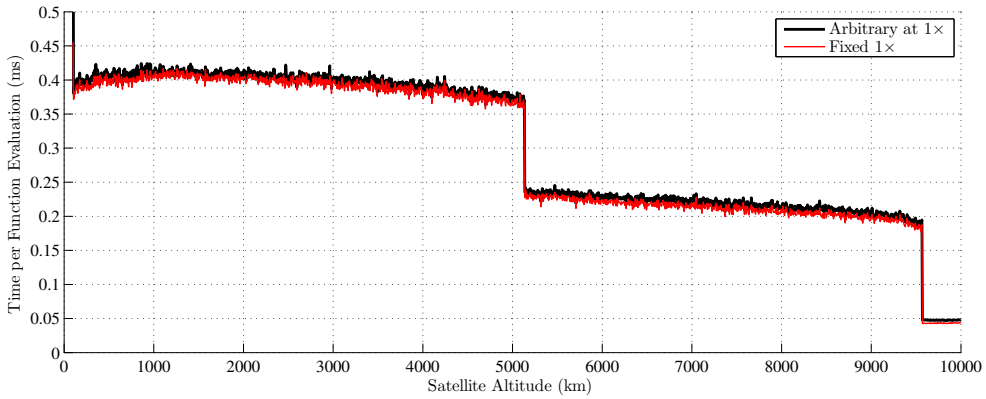
Previous discussions on performance (Section 3.1.6) are focused specifically on the underlying clipping implementations. In this section, runtimes of the actual implemented coverage models (as described in Section 3.2) are compared.

3.4.1 Arbitrary vs. Fixed Coverage Algorithms

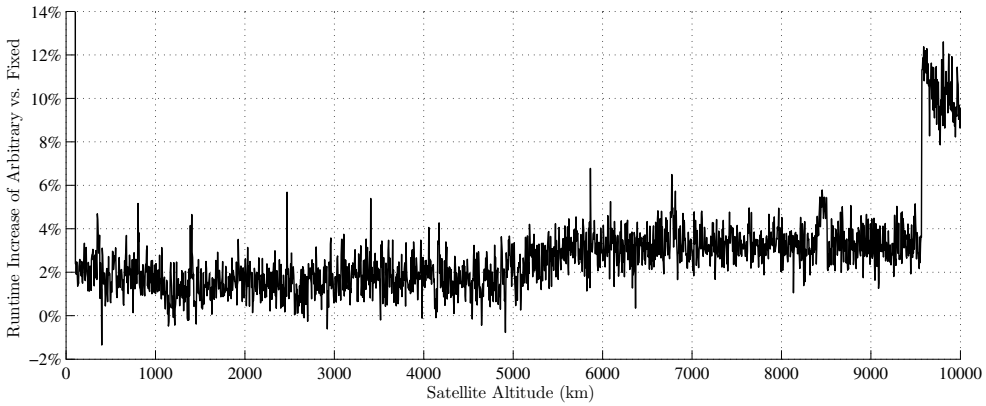
As discussed at the beginning of Section 3.2.5, regions of single, double, and triple coverage can either be computed using the ATH coverage models specifically tailored to each coverage multiplicity (Sections 3.2.2 through 3.2.4), or using the

arbitrary coverage multiplicity model (Section 3.2.5). However, due to the additional provisions to ensure generality, it is expected that the arbitrary coverage multiplicity model executes slower than its fixed-multiplicity counterparts.

First, consider re-solving Marchand and Kobel's[4] 'Example 1' discussed in Section 3.1.7. Time per function evaluation is compared between the single coverage model described in Section 3.2.2 and the arbitrary coverage multiplicity model (analyzing for single coverage) described in Section 3.2.5. Comparisons are performed using coverage model implementations in C++ using the GPC library, discussed in Section 3.1.1. The results are shown in Figure 3.33 – clearly the arbitrary coverage model, as expected, takes slightly longer per function evaluation. Figure 3.33b shows this runtime increase (relative to the fixed single coverage model) to be typically on the order of 2-3%.



(a) Time per Function Evaluation for the Fixed 1× vs. Arbitrary at 1× ATH Coverage Models

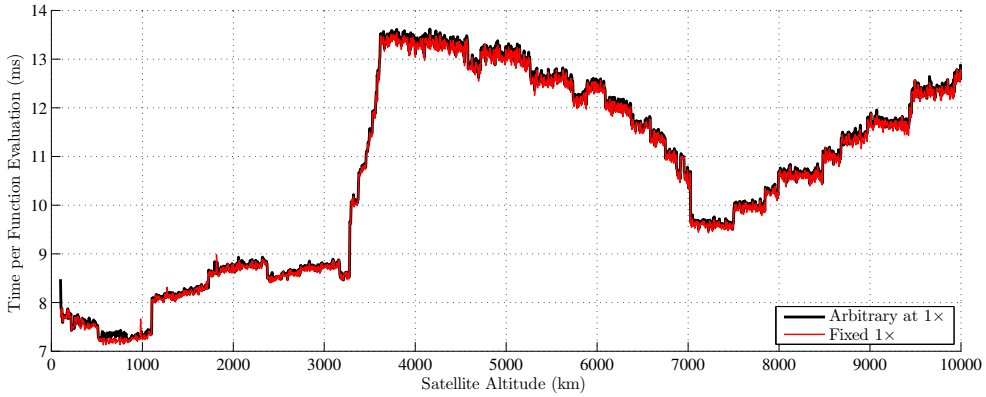


(b) Percent Runtime Increase of Arbitrary at 1× vs. Fixed 1× ATH Coverage Model (GPC/C++)

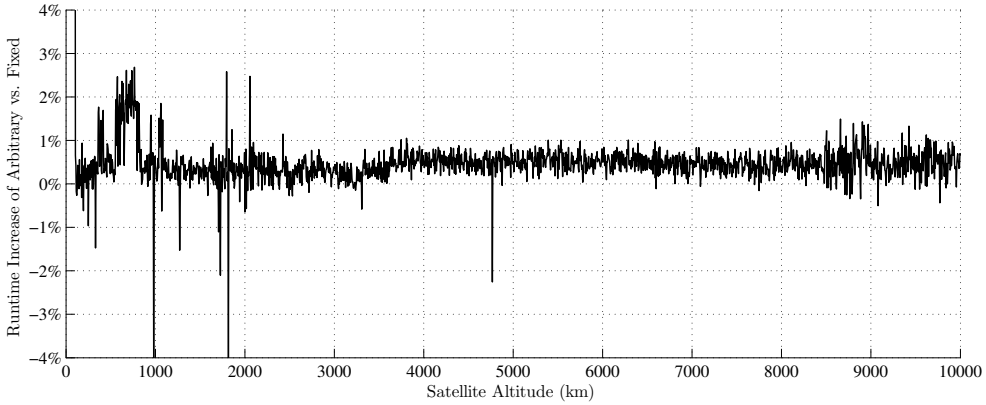
Figure 3.33: Single Satellite Case – Fixed vs. Arbitrary Multiplicity Coverage Models (Single Coverage)

Next, consider the 12 satellite problem, analyzed for error in Section 3.3.3. Figure 3.34 shows the performance comparison between fixed and arbitrary multiplicity coverage models for single coverage. Similarly, the arbitrary multiplicity coverage model is tested against the fixed double and triple coverage models, with results shown in Figures 3.35 and 3.36. In all cases, the fixed multiplicity cover-

age models exhibit slightly superior performance, with the arbitrary multiplicity coverage model typically running 1-2% slower. The regions of each curve where runtime gradually rises and falls are due to the increase and decrease of intermediate polygon size for that particular regime. Discontinuities are due to the analysis entering regions where the clipping implementation can perform shortcuts, avoiding unnecessary clipping operations.

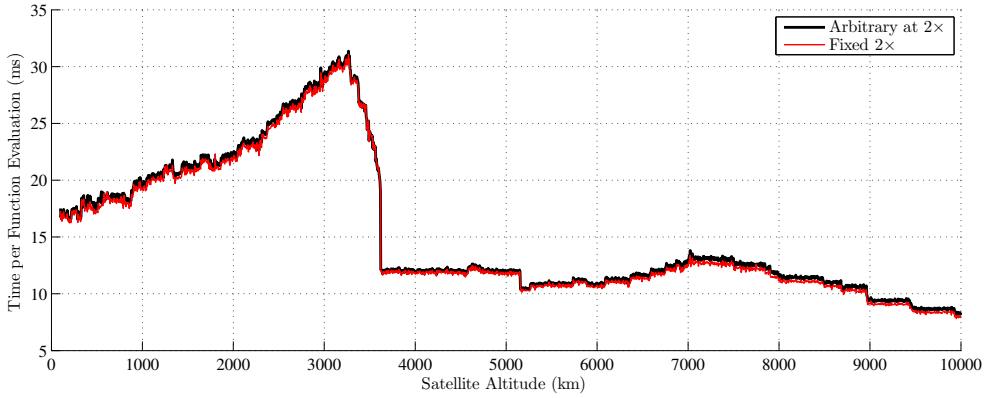


(a) Time per Function Evaluation for the Fixed 1× vs. Arbitrary at 1× Coverage Models

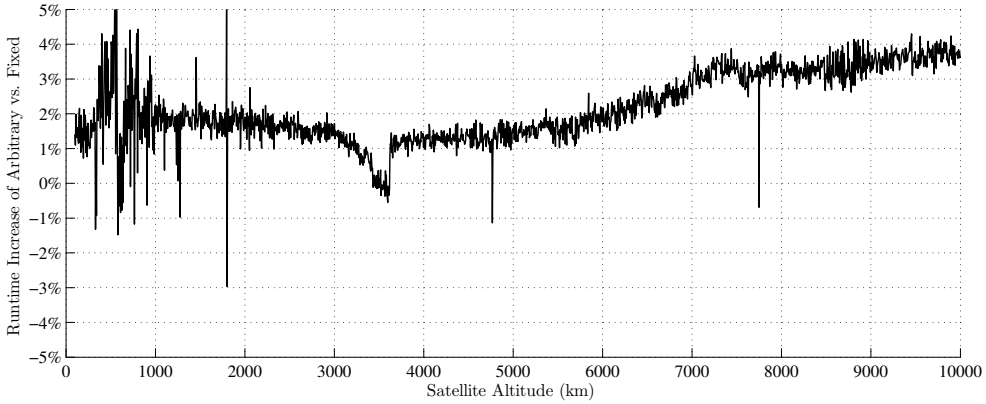


(b) Percent Runtime Increase of Arbitrary at 1× vs. Fixed 1× Coverage Model (GPC/C++)

Figure 3.34: 12 Satellite Case – Fixed vs. Arbitrary Multiplicity Coverage Models (Single Coverage)

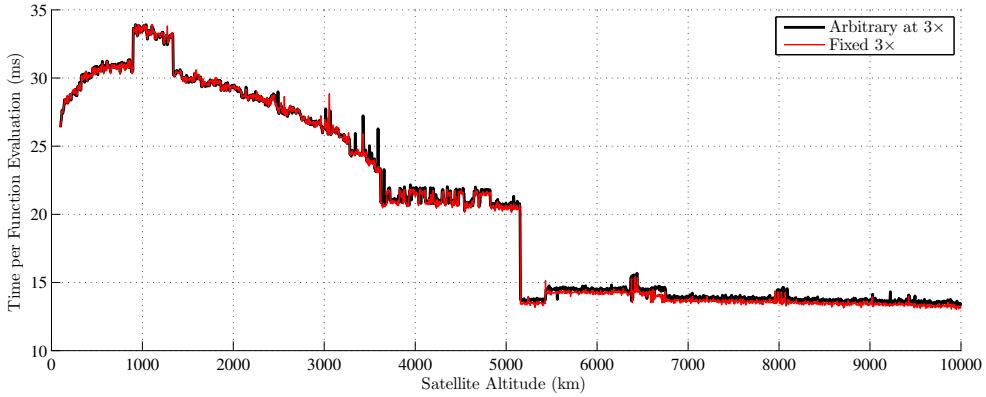


(a) Time per Function Evaluation for the Fixed 2× vs. Arbitrary at 1× Coverage Models

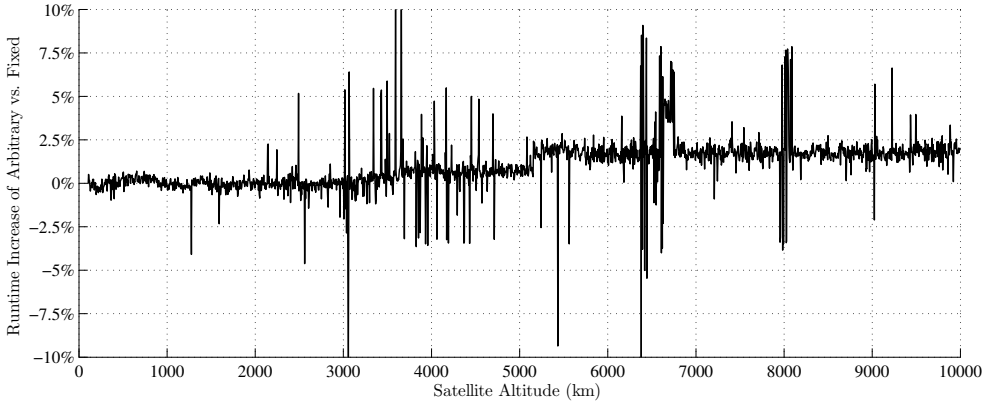


(b) Percent Runtime Increase of Arbitrary at 2× vs. Fixed 1× Coverage Model (GPC/C++)

Figure 3.35: 12 Satellite Case – Fixed vs. Arbitrary Multiplicity Coverage Models (Double Coverage)



(a) Time per Function Evaluation for the Fixed 3× vs. Arbitrary at 1× Coverage Models



(b) Percent Runtime Increase of Arbitrary at 3× vs. Fixed 1× Coverage Model (GPC/C++)

Figure 3.36: 12 Satellite Case – Fixed vs. Arbitrary Multiplicity Coverage Models (Triple Coverage)

Chapter 4

Example Problems

To illustrate the utility of the numerical ATH coverage models, several example problems are solved. Prior to this, a simple continuously differentiable financial model describing constellation deployment cost is developed for use as both an objective function and a constraint function. Example 1 considers maximization of single coverage by a constellation subject to a deployment cost constraint. Example 2 explores the reverse case: a minimum acceptable ATH coverage amount is specified while deployment cost is minimized. Example 3 utilizes the numerical ATH coverage models as both objective and constraint functions. A minimum amount of single coverage, and a maximum deployment cost are prescribed, while maximizing double ATH coverage. Finally, Example 4 illustrates the use of an arbitrary sensor profile in a constellation design problem, maximizing single ATH coverage subject to a constraint on deployment cost.

4.1 Simple Financial Model

The design problems described in this chapter involve constraining or minimizing financial cost, while simultaneously maximizing or constraining (respectively) some amount of coverage at a specified coverage multiplicity. Models estimating program cost for a satellite or constellation are used throughout government and

industry (see Wertz).[8] These models are highly complex with a staggering amount of parameters involved. Additionally, these exhaustive models can behave discontinuously in many regions, i.e. when a payload of several satellites becomes too massive for one launch vehicle, necessitating another.

The objective of the examples in this chapter is not to illustrate these complex cost estimation relations, but to illustrate how the numerical ATH coverage models described in Chapters 2 and 3 can be used as part of a constellation design process. Consequently, a much simpler financial model is necessary, one that is continuous, easily described to the reader, and easily implemented by the investigator.

The model developed here focuses on two parameters as cost drivers behind constellation deployment – the mass of each spacecraft, and their circular orbit altitude. As discussed previously, this research focuses on constellations composed of satellites arranged in a single circular orbit. The time-invariant nature of this set of problems allows them to be far more tractable using the limited computational resources available.

4.1.1 Constellation Deployment Cost

According to Gordon,[12] the mass of a sensor or antenna can be crudely described as proportional to the square of its design range. Using this relationship, spacecraft mass can be approximated by

$$m = a + bR^2. \tag{4.1}$$

The parameter a is the base satellite mass (i.e. if no sensor were installed at all), and b represents the mass/sensor-range coefficient. The total cost of constellation

deployment is then described by

$$\Gamma = n(e + cm + dm(h - h_{\text{ref}})^2). \tag{4.2}$$

A linear relationship between the number of satellites, n, and deployment cost Γ is assumed. The parameter e represents a base cost for each launch system, c represents the additional cost per unit mass to achieve the reference circular orbit at an altitude of h_{ref}. The cost of increasing satellite altitude (h) from the reference altitude is assumed to increase quadratically, thus the coefficient d describes the cost per unit mass vs. the square of the increase in altitude. This assumption of quadratic behavior with respect to altitude variation is made to create a more interesting deployment cost model, i.e. not just quadratic in R, but also in h. In all analyses presented here, h_{ref} is considered to be the lower bound on permissible satellite altitude.

Mass and altitude are the originally intended cost drivers; however, by Equation 4.1, R is used in place of mass as a model parameter.

4.1.2 Derivatives

Although analytical derivatives for numerically evaluated ATH coverage are not available, the partial derivatives of the simple financial model defined in Section 4.1.1 are obtained easily. Substituting Equation 4.1 into Equation 4.2 and simplyfing

yields

$$\Gamma = n(e + c(a + bR^2) + d(a + bR^2)(h - h_{\text{ref}})^2) \tag{4.3}$$

$$= n(e + ca + cbR^2 + (da + dbR^2)(h - h_{\text{ref}})^2) \tag{4.4}$$

$$= ne + nca + ncbR^2 + nda(h - h_{\text{ref}})^2 + ndbR^2(h - h_{\text{ref}})^2. \tag{4.5}$$

Partial derivatives of Equation 4.5 in R and h are then evaluated, producing

$$\frac{\partial \Gamma}{\partial R} = 2nb(cR + 2ndR(h - h_{\text{ref}})^2), \tag{4.6}$$

$$\frac{\partial \Gamma}{\partial h} = 2nd(a + bR^2)(h - h_{\text{ref}}). \tag{4.7}$$

4.1.3 Parameter Values

The same set of financial model parameters is used in Examples 1 through 3. Example 4 considers a satellite sensor profile of arbitrary shape, where the mass/sensor-range coefficient is reduced by a factor of four. This assumption is made because, as will be seen in Section 4.6, the arbitrary region encompassed in RS_{arb} is much smaller than in an omni-directional case using the same value of R. The resulting financial costs computed using these parameters are not necessarily correlated to real world dollars, and are only intended as a means of comparison between the solutions and problems presented here. They are, however, selected with the aim of obtaining results on the same order-of-magnitude as a more exhaustive analysis.[8] These parameters are summarized in Table 4.1.

Table 4.1: Financial Model Parameters (see Equations 4.1, 4.2)

Parameter	Value	Description
a	500 kg	base satellite mass
b_{OD}	6×10^{-5} kg/km^2	mass/sens-range coefficient (OD sensors)
b_{arb}	1.5×10^{-5} kg/km^2	mass/sens-range coefficient (arb. sensors)
c	1×10^{-4} \$M/kg	cost-per-kg at reference altitude
d	1×10^{-9} \$M/kg-km^2	cost-per-kg vs. squared altitude increase
e	10 \$M	base launch vehicle cost
h_{ref}	188 km	satellite reference altitude

4.2 Non-Linear Programming

The examples presented in this chapter are simple constellation design problems, and their solutions are easily obtained by parameter optimization. The problems are parameterized in terms of a finite set of variables that fully define a unique state of the system. The solution, or optimal values of these variables create some optimal condition of the system in whatever sense the investigator defines (by finding the states providing minimal or maximal values of some performance index). Although the examples in this chapter are simple enough to be analyzed based on clever plots alone, the same method of parameter optimization can be readily scaled to more complex problems, for which visualization of the phase space becomes impossible.

The parameter optimization code used here is provided in the MATLAB Optimization Toolbox – *fmincon*,[29] which is a package of several different non-linear programming (NLP) packages. Loosely defined, NLP is a process by which some objective function is extremized subject to equality and/or inequality constraints, all dependent upon a finite set of optimization variables. Additionally, the objective

function and the constraints could both exhibit nonlinear behavior (NLP can be considered a superset of linear programming – linear programming can be used only when the objective and constraint functions are linear). Specifically, the sequential quadratic programming (SQP) algorithm (see Powell)[30] in *fmincon* is used to obtain solutions to the example problems in this chapter. The general problem solved by NLP is as follows:

$$\text{minimize } J = F(\mathbf{x}_p), \tag{4.8}$$

$$\text{subject to } \mathbf{c}_{\text{eq}}(\mathbf{x}_p) = \mathbf{0}, \tag{4.9}$$

$$\mathbf{c}(\mathbf{x}_p) \leq \mathbf{0}, \tag{4.10}$$

keeping in mind that maximizing a function is equivalent to minimizing its negative.

4.3 Example 1 – Max. Single Coverage with a Budget Constraint

Consider a constellation, constrained by deployment cost (as defined by the financial model in Section 4.1) that is desired to provide maximum possible single coverage within the dual-altitude band area of interest. The parameters describing the tangent height shell (THS) and the dual-altitude band shell, AS, are shown in Table 4.2. Using h_l and h_u at $m = 100$ PPC, the area inside the altitude shell, A_{AS}, is also computed and listed at the end of the table.

It is assumed that the satellites are equally distributed in a single circular orbit, and possess omni-directional sensors. Under these assumptions, the problem can be uniquely described by two optimization parameters – satellite sensor range, R, and satellite altitude, h. This two-dimensional set of variables allows straightforward

Table 4.2: Example 1 Parameters

Parameter	Value	Description
R_E	6378.14 km	assumed Earth radius
h_t	100 km	tangent height altitude
h_l	1000 km	lower target altitude
h_u	5000 km	upper target altitude
m	100 PPC	initial polygon resolution
A_{AS}	235,307,769 km^2	total area in AS

verification and discussion of results using plots. Thus, the parameter vector is formed as

$$\mathbf{x}_p = [R \quad h]. \tag{4.11}$$

The performance index (single coverage area) is given by

$$J = -A_{1\times}(R, h, n), \tag{4.12}$$

with a minus sign introduced so that minimization of J is equivalent to maximization of $A_{1\times}$ (this conforms to standard convention for most NLP codes of minimization of functions rather than explicit maximization). Analytical derivatives of J are not available simply because the relationships between $A_{1\times}$, R, and h are purely numerical in nature.

Three inequality constraints are present – first, the cost must be less than the deployment budget, Γ_{max}. Second, the sensor range must be non-negative (i.e. its negative must be less than zero). Finally, the satellite circular orbit altitude must be above the reference altitude, h_{ref}. These constraints can be written in vector form

as

$$\mathbf{c} = \begin{bmatrix} \Gamma(R, h, n) - \Gamma_{\max} \\ -R \\ h_{\text{ref}} - h \end{bmatrix} \leq \mathbf{0}, \tag{4.13}$$

where the inequality is understood to apply element-wise to the elements of $\mathbf{c}$. Analytical derivatives of the inequality constraints are available, and are tested for each constellation size to establish their validity. However, for this example, finite difference derivatives lead to convergence in far fewer iterations, and cause an unnoticeable increase in runtime.

First, a single satellite is considered with a modest deployment budget. For comparison, a 10 satellite constellation is then analyzed, subject to a 10-fold increase in deployment budget. A 15 satellite constellation is investigated with a 40% increase in deployment budget over the 10 satellite case, but the results are omitted from this chapter for brevity – they are instead presented in Appendix B.1.

4.3.1 Example 1a – 1 Satellite

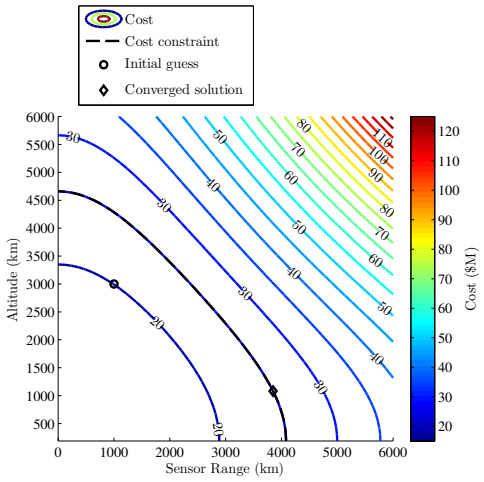
The single satellite case can be considered a simple extension of the problem presented by Marchand and Kobel,[4] except omni-directional sensor range is now an optimization parameter, and deployment cost is now considered and constrained. The deployment budget and initial guess are presented along with the converged solution providing maximum single coverage subject to the deployment budget in Table 4.3. Budget overrun is a measure of deployment cost constraint violation at the converged solution.

The financial model behavior and single coverage area vs. R and h are shown in Figures 4.1a and 4.1b respectively. Figure 4.1c shows the initial guess and con-

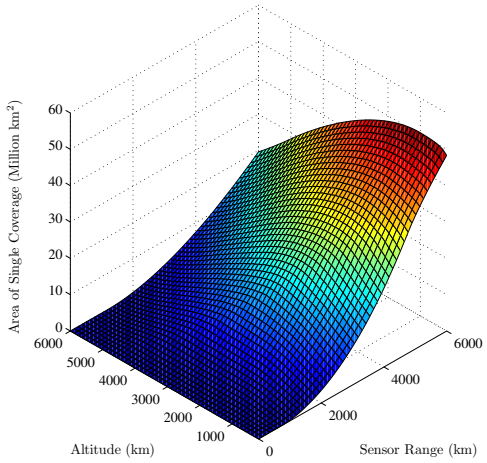
verged solutions, along with the cost constraint overlaid on the single coverage area contour. It is clear from this plot that the converged solution corresponds to a constrained local maximum in coverage area, as expected. The coverage configurations for the initial guess and converged solution are illustrated in Figure 4.2.

Table 4.3: Example 1a – Parameters and Solution

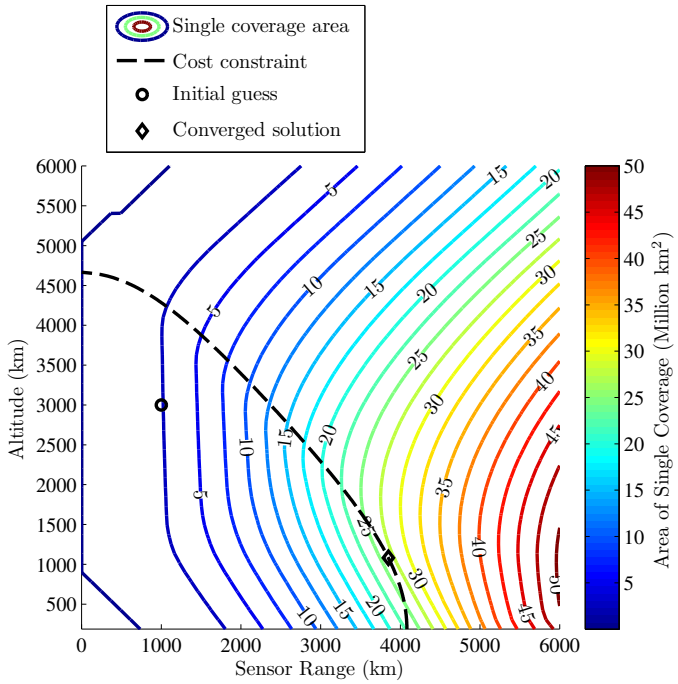
Parameter	Value	Description
n	1	number of satellites
$\Gamma_{\max}$	\$25M	budget constraint
R_0	1000 km	initial guess, R
h_0	3000 km	initial guess, h
R_{opt}	3849.844 km	converged area-optimal R
h_{opt}	1080.735 km	converged area-optimal h
$A_{1\times_{\text{opt}}}$	$26,473,840$ km^2	coverage area at solution
$\Gamma_{\text{opt}} - \Gamma_{\max}$	9.948×10^{-9} \$M	budget overrun



(a) Constraint Function (Financial Cost)

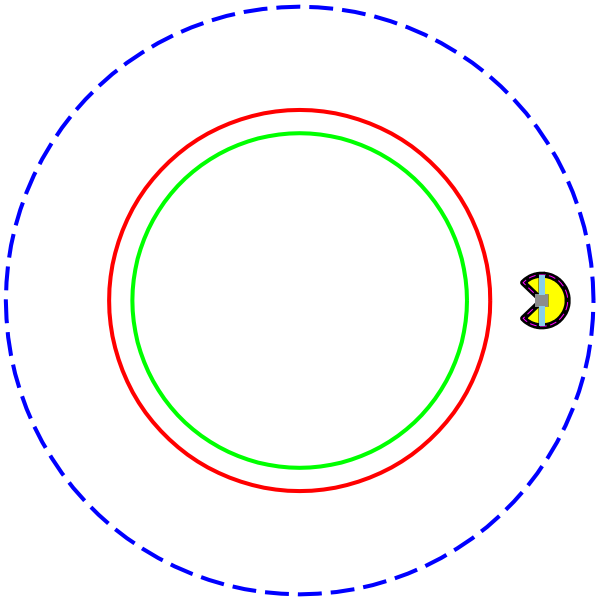


(b) Objective Function (Single Coverage Area)

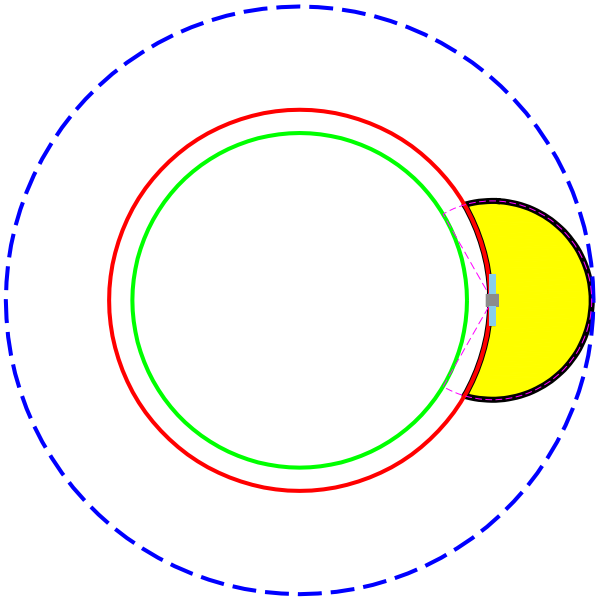


(c) Objective Function (Single Coverage Area) With Converged Solution

Figure 4.1: Example 1a – Constraint and Objective Contours



(a) Initial Guess



(b) Converged Solution

Figure 4.2: Example 1a – Initial Guess and Converged Solution

4.3.2 Example 1b – 10 Satellites

The 10 satellite case is given a 10 fold increase in budget relative to the single satellite case discussed in Example 1a. However, the maximum possible single coverage does necessarily scale to 10 times the single satellite case. The possibility for overlap of sensor regions (if the budget allows for a sufficient R) means that some regions will exhibit single coverage redundantly (i.e. double coverage). The initial guess, budget constraint and converged solution are shown in Table 4.4.

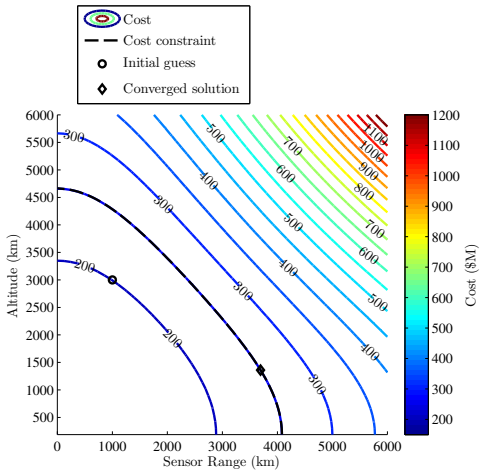
Table 4.4: Example 1b – Parameters and Solution

Parameter	Value	Description
n	10	number of satellites
$\Gamma_{\max}$	\$250M	budget constraint
R_0	1000 km	initial guess, R
h_0	3000 km	initial guess, h
R_{opt}	3694.174 km	converged area-optimal R
h_{opt}	1360.112 km	converged area-optimal h
$A_{1\times_{\text{opt}}}$	$208,625,437$ km^2	coverage area at solution
$\Gamma_{\text{opt}} - \Gamma_{\max}$	6.314×10^{-10} \$M	budget overrun

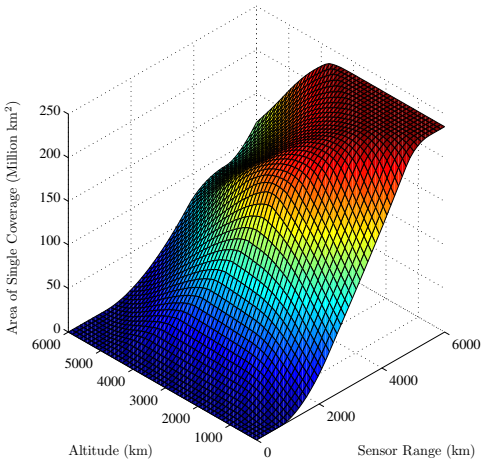
As expected, despite a 10 fold increase in budget, the area at the converged solution (Table 4.4) is not even close to 10 times the single coverage area found in the single satellite case (Table 4.3). The primary reason for this is clearly illustrated in Figure 4.4b – note the overlap between the sensor range shells (dashed regions centered on each satellite) in the converged solution. These regions exhibit redundant single coverage (i.e. double coverage), and are a 'waste' of coverage given that only single coverage is desired in this particular problem.

Figure 4.3 shows that the budget constraint allows coverage approaching

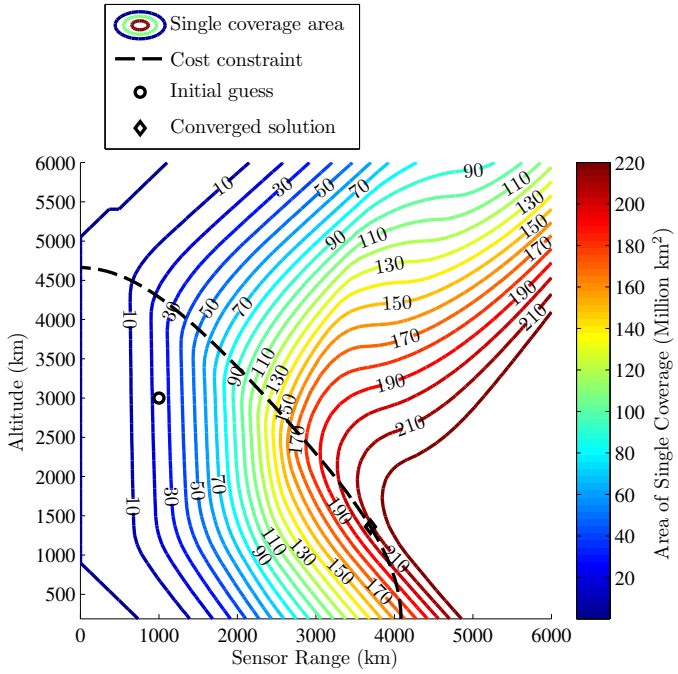
saturation of the area of interest. In fact, the 208 million km^2 of single coverage at the solution corresponds to approximately 89% of the dual-altitude band shell. However, a budget that is significantly larger could extend the cost constraint curve onto the level plateau in Figure 4.3c, where single coverage is saturated for a variety of R and h values. The optimization problem then becomes ill-posed, because an infinite number of solutions exist in the plateau region that result in the same coverage area, while simultaneously satisfying the budget constraint.



(a) Constraint Function (Financial Cost)

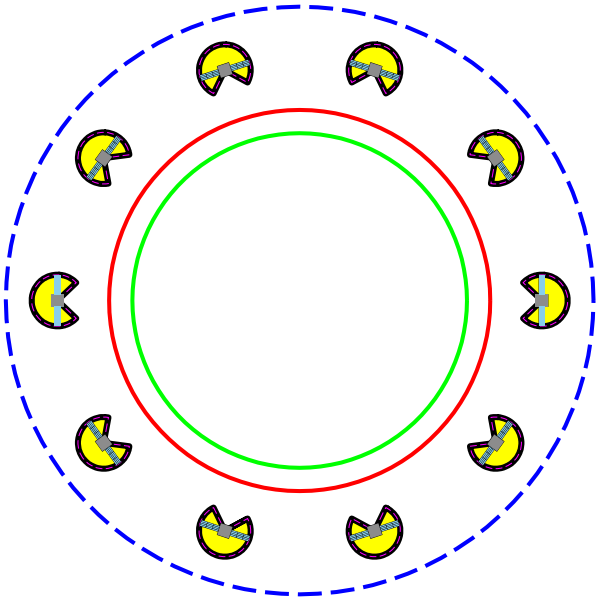


(b) Objective Function (Single Coverage Area)

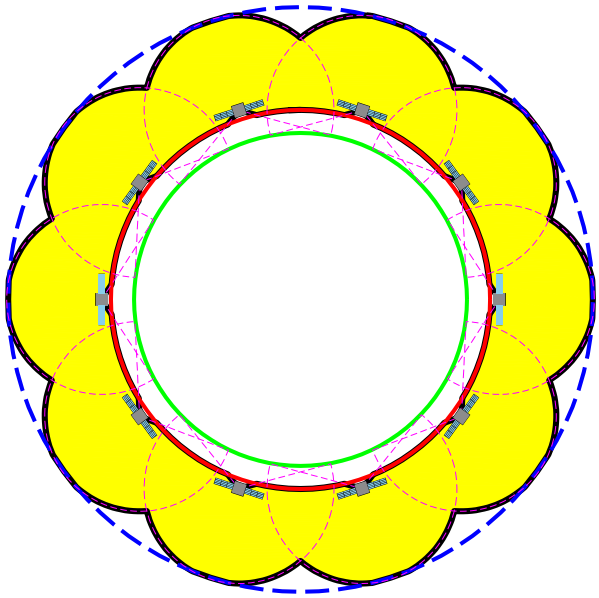


(c) Objective Function (Single Coverage Area) With Converged Solution

Figure 4.3: Example 1b – Constraint and Objective Contours



(a) Initial Guess



(b) Converged Solution

Figure 4.4: Example 1b – Initial Guess and Converged Solution

4.4 Example 2 – Min. Budget with an Area Constraint

The constellations designed in this example must provide single coverage to at least 99.9% of the total area enclosed in the dual-altitude band shell. This value, rather than 100%, is arbitrarily chosen to avoid numerical issues associated with roundoff and truncation error (however small they may be). The area of the altitude shell, A_{AS}, is computed using the same polygon resolution as the altitude shell considered in each problem. The parameters describing the tangent height shell (THS) and the dual-altitude band shell, AS, are shown in Table 4.5 along with the computed area inside AS.

Table 4.5: Example 2 Parameters

Parameter	Value	Description
R_E	6378.14 km	assumed Earth radius
h_t	100 km	tangent height altitude
h_l	1000 km	lower target altitude
h_u	5000 km	upper target altitude
m	100 PPC	initial polygon resolution
A_{AS}	235,307,769 km^2	total area in AS

As with Example 1, assume that the satellites are equally distributed in a single circular orbit, and are equipped with omni-directional sensors. Under these assumptions, the satellite sensor range, R, and the satellite altitude, h, are sufficient to fully define a unique state of the system. The two-dimensional set of variables populates the parameter vector

$$\mathbf{x}_p = \begin{bmatrix} R & h \end{bmatrix}. \tag{4.14}$$

The performance index that is minimized with respect to $\mathbf{x}_p$ (deployment

cost, according to the model developed in Section 4.1) is given by

$$J = \Gamma(R, h, n). \tag{4.15}$$

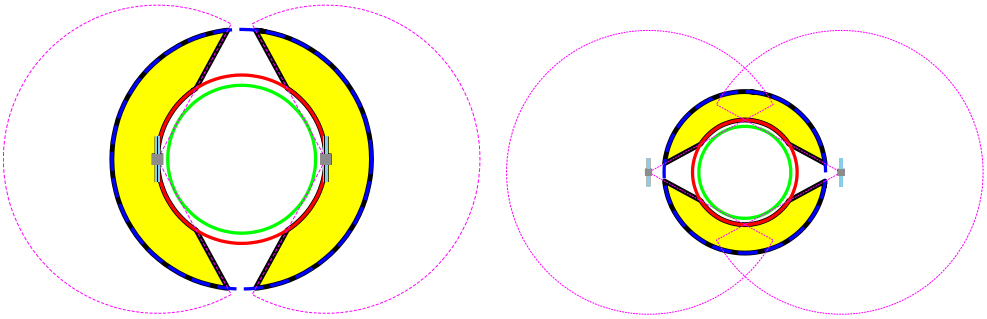
Additionally, three inequality constraints are present as follows:

$$\mathbf{c} = \begin{bmatrix} A_{\min} - A_{1\times} \\ -R \\ h_{\text{ref}} - h \end{bmatrix} \leq \mathbf{0}, \tag{4.16}$$

where the inequality is understood to apply element-wise between vectors $\mathbf{c}$ and $\mathbf{0}$.

While many cases are investigated in this example, only three are presented here, for brevity. Discussion, however, is not limited only to those three – plots and solutions for the remaining cases are included in Appendix B.

A three satellite constellation is the smallest constellation that can achieve the desired coverage amount for the dual-altitude band shell described in Table 4.5. The coverage constraint requires at least 99.9% single coverage of AS, but Figure 4.5 clearly shows this is impossible with only two satellites. In Figure 4.5a, the satellite altitude is low enough to bring the tip of the tangent height triangle (THT) below the lower target altitude shell (LTAS), but the lower target altitude is low enough that this altitude results in large coverage gaps, as shown at the top and bottom of AS. The opposite scenario is shown in Figure 4.5b, where the satellite altitude and range are sufficient to close the gaps at the top and bottom of AS, but result in a very large coverage gap in the nadir direction of each satellite due to the THT.



(a) $R = 13500$ km, $h = 1000$ km (b) $R = 20000$ km, $h = 7200$ km

Figure 4.5: Two Satellites – Total Single Coverage is Impossible (for the Parameters in Table 4.5)

The three satellite case is analyzed and discussed first – the smallest possible constellation for the problem parameters chosen. Next, the four satellite case presents an interesting scenario with multiple local maxima in coverage area. The 10 satellite case is then investigated, which proves to be the largest and most costly in terms of deployment cost. Finally, constellations consisting of 5 through 9 satellites are analyzed, and a trend in minimum constellation deployment cost is discussed, concluding that the lower-cost four satellite configuration is the most cost-efficient using the financial model developed in Section 4.1.

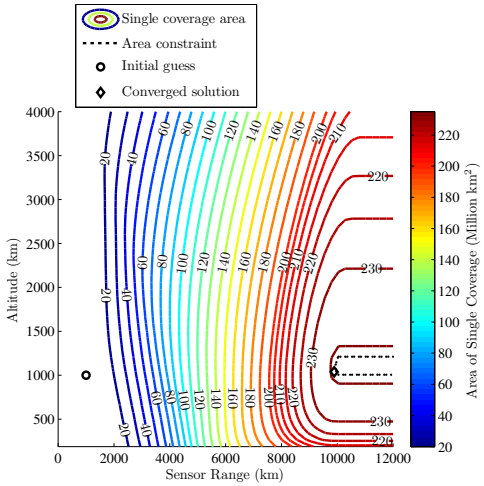
4.4.1 Example 2a – 3 Satellites

The parameters and solution to the 3 satellite case are shown in Table 4.6. Comparing this result to the solution found in Example 1b, in Section 4.3.2, yields an interesting difference – for a deployment cost of only 235 $M, three satellites cover far more of AS than 10 satellites (which cost 250 $M to deploy).

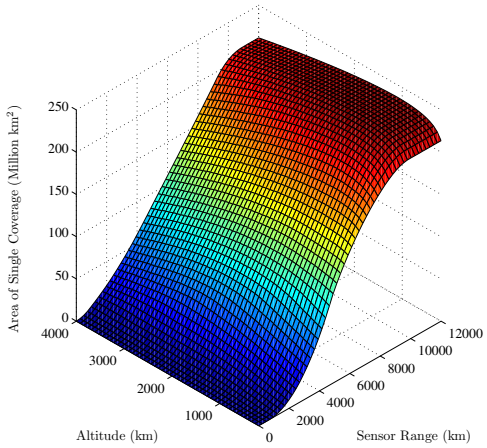
Figure 4.6a shows the area constraint (the solution must lie inside or on the dotted boundary). From inspection of Figure 4.6c, it is clear to see that the converged solution does correspond to a constrained minimum in deployment cost. The configuration of the converged solution, shown in Figure 4.7b, illustrates that the sensor range corresponds to the intersection between neighboring satellite sensor range shells and the upper target altitude shell (UTAS), as expected by intuition.

Table 4.6: Example 2a – Parameters and Solution

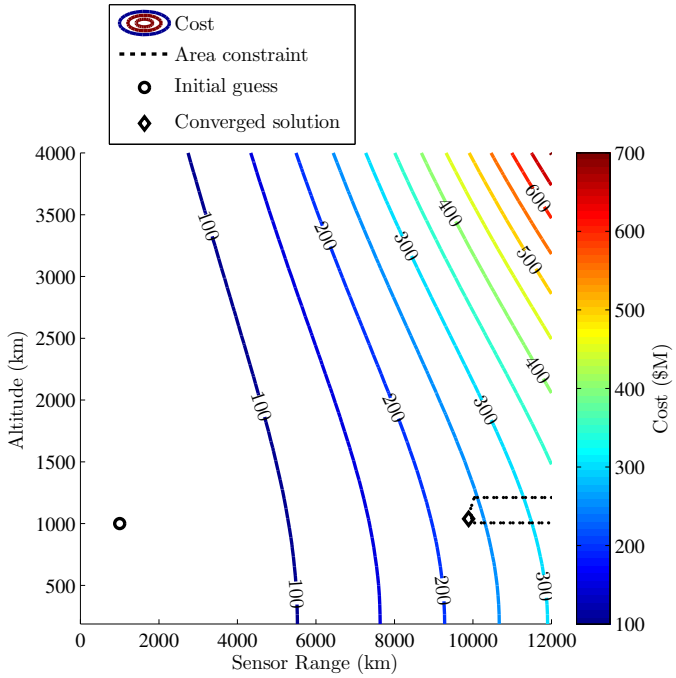
Parameter	Value	Description
n	3	number of satellites
$A_{\min}$	$0.999A_{AS}$	area constraint
R_0	1000 km	initial guess, R
h_0	1000 km	initial guess, h
R_{opt}	9889.940 km	converged cost-optimal R
h_{opt}	1038.909 km	converged cost-optimal h
Γ_{opt}	234.893 $M	converged cost at solution
$A_{1\times_{\text{opt}}} - A_{\min}$	-1.095×10^{-2} km^2	coverage area surplus



(a) Constraint Function (Single Coverage Area)

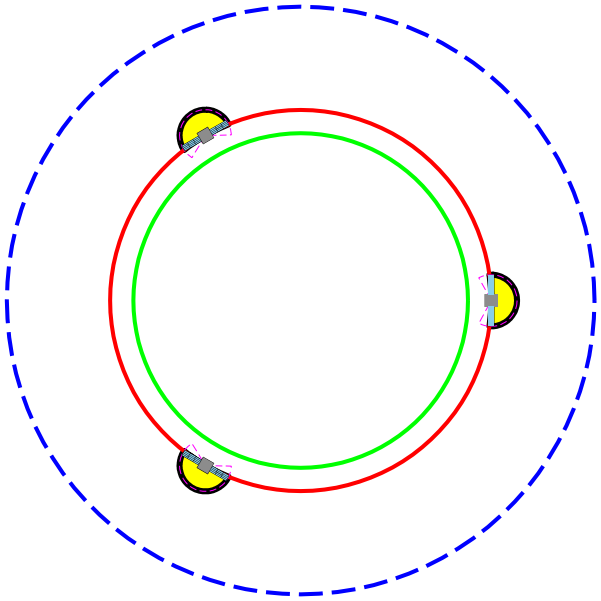


(b) Constraint Function (Single Coverage Area)

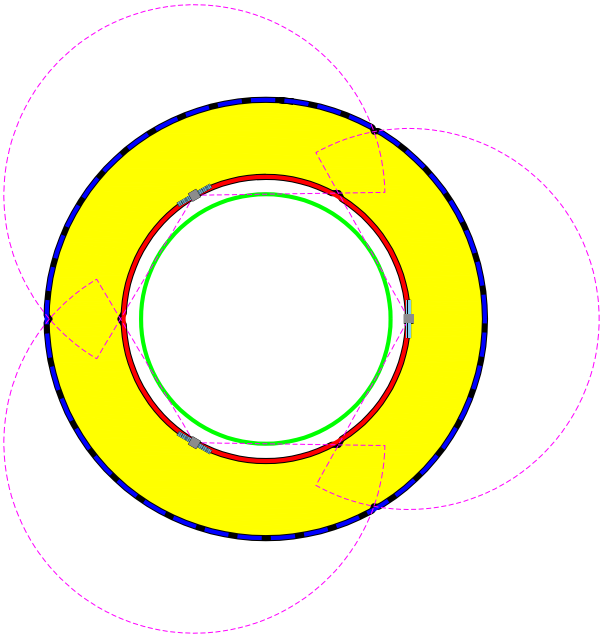


(c) Objective Function (Financial Cost)

Figure 4.6: Example 2a – Constraint and Objective Contours



(a) Initial Guess



(b) Converged Solution

Figure 4.7: Example 2a – Initial Guess and Converged Solution

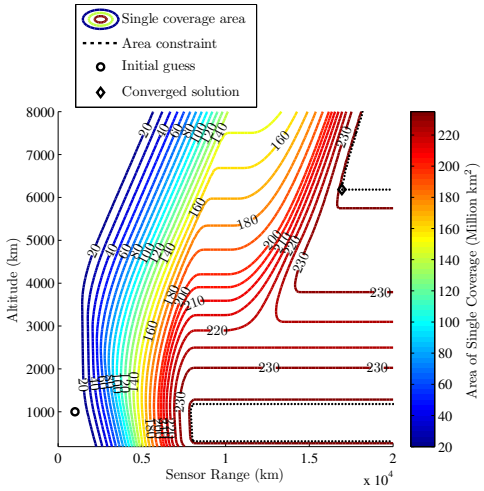
4.4.2 Example 2b – 4 Satellites

The four satellite case is interesting in that it exhibits two disparate regions in R and h that both satisfy the area constraint. Using the initial guess shown in Table 4.7, the optimizer finds a constrained local minimum in deployment cost with a very large sensor range, and a very high altitude. Because sensor range (and consequently mass) and satellite altitude are the cost drivers in the financial model developed in Section 4.1, these large values result in a 3.3 $B budget requirement – an unacceptable result.

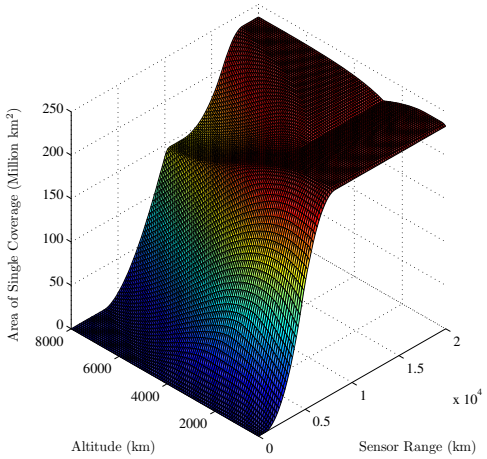
Consider Figure 4.8 – the optimizer traversed the domain to the more distant constrained local minimum, that is clearly not a global minimum from inspection of the plots. Figure 4.9b shows that this near-total coverage is achieved when neighboring satellites fill the coverage gap in the nadir direction of each satellite.

Table 4.7: Example 2b – Parameters and Solution (Case 1)

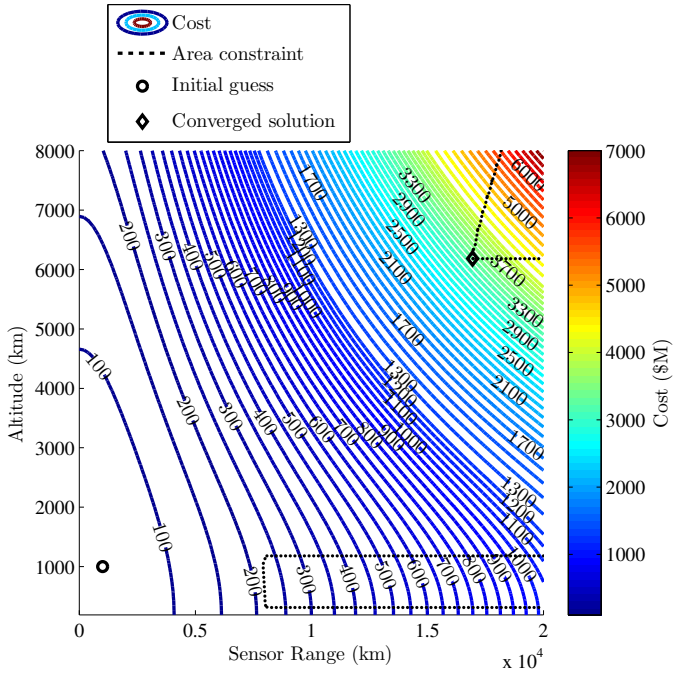
Parameter	Value	Description
n	4	number of satellites
$A_{\min}$	$0.999A_{AS}$	area constraint
R_0	1000 km	initial guess, R
h_0	1000 km	initial guess, h
R_{opt}	16943.198 km	converged cost-optimal R
h_{opt}	6184.123 km	converged cost-optimal h
Γ_{opt}	3297.977 $M	converged cost at solution
$A_{1\times_{\text{opt}}} - A_{\min}$	-9.924×10^{-6} km^2	coverage area surplus



(a) Constraint Function (Single Coverage Area)

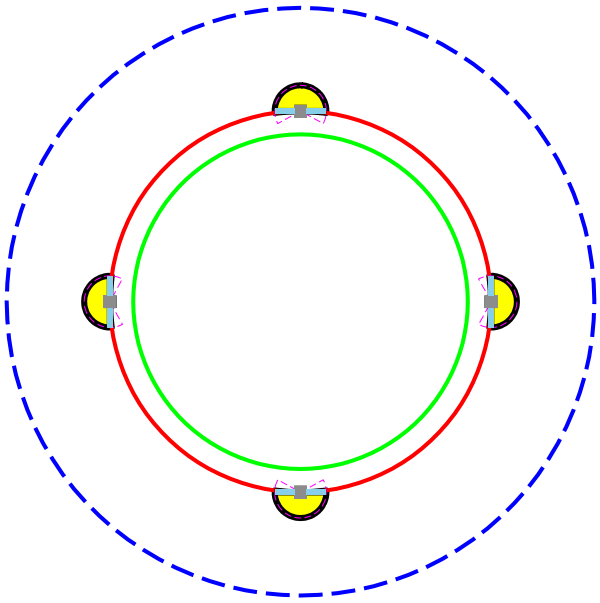


(b) Constraint Function (Single Coverage Area)

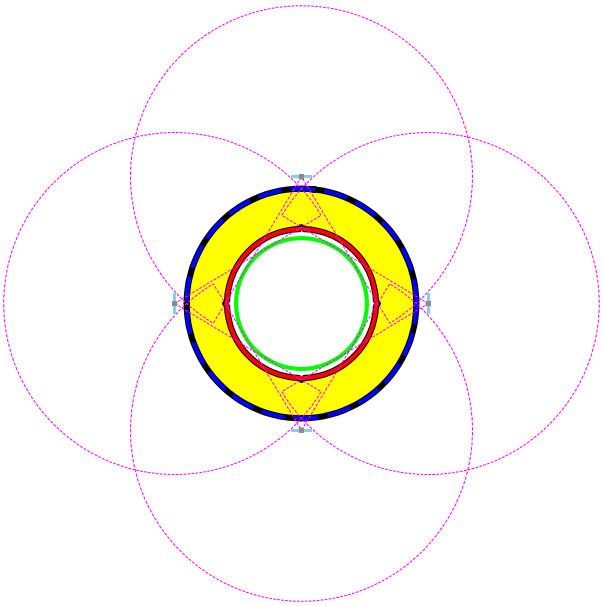


(c) Objective Function (Financial Cost)

Figure 4.8: Example 2b – Constraint and Objective Contours (Case 1)



(a) Initial Guess



(b) Converged Solution

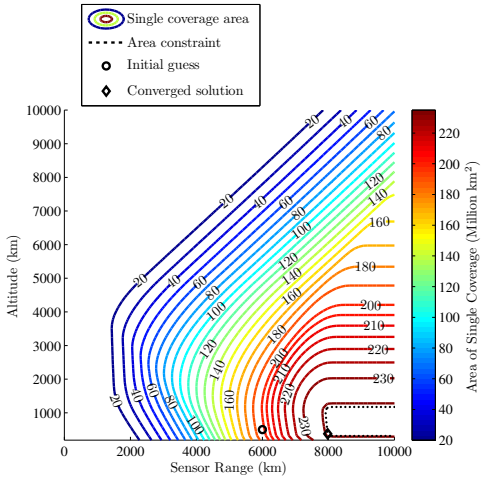
Figure 4.9: Example 2b – Initial Guess and Converged Solution (Case 1)

The second solution, resulting from an initial guess closer to the desired local minimum, is far more reasonable, as shown in Table 4.8. Satellite altitude is low, and sensor range is on par with the 3 satellite case, as expected. The deployment budget of only 214 $M turns out to be the most cost-efficient constellation analyzed in this example to deliver near-total single coverage in AS.

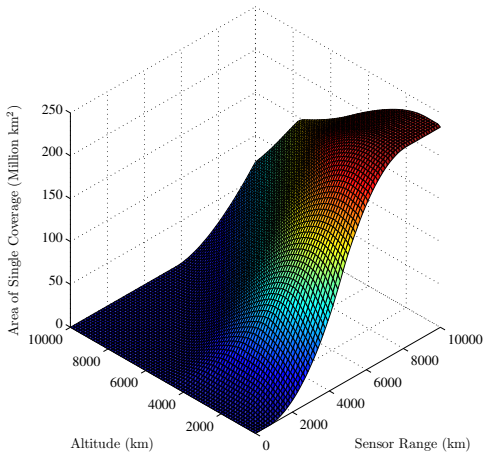
Figure 4.10 illustrates that the converged solution is a constrained local minimum, as desired (note that the upper bound on sensor range is different from that in Figure 4.8). Figure 4.11b illustrates again that the optimal sensor range causes an intersection between neighboring range shells with the upper target altitude shell (UTAS), as expected.

Table 4.8: Example 2b – Parameters and Solution (Case 2)

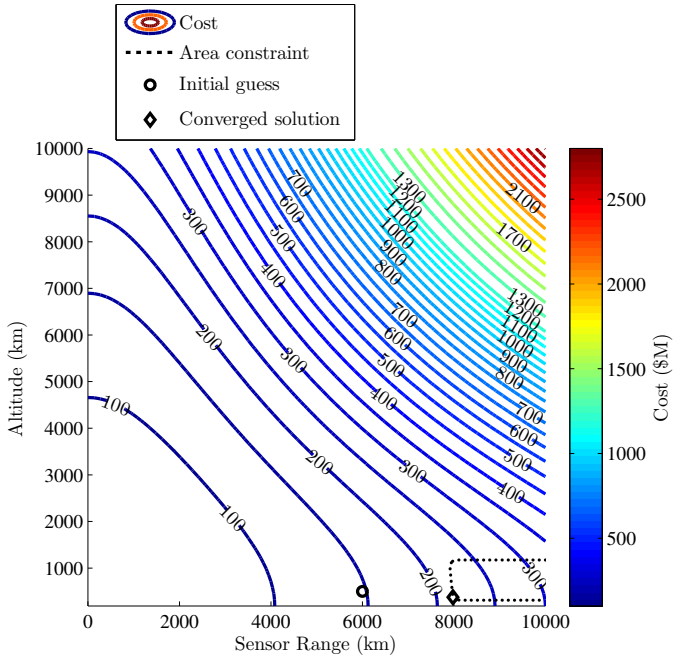
Parameter	Value	Description
n	4	number of satellites
$A_{\min}$	$0.999A_{AS}$	area constraint
R_0	6000 km	initial guess, R
h_0	500 km	initial guess, h
R_{opt}	7978.556 km	converged cost-optimal R
h_{opt}	376.418 km	converged cost-optimal h
Γ_{opt}	213.391 $M	converged cost at solution
$A_{1\times_{\text{opt}}} - A_{\min}$	1.167×10^{-4} km^2	coverage area surplus



(a) Constraint Function (Single Coverage Area)

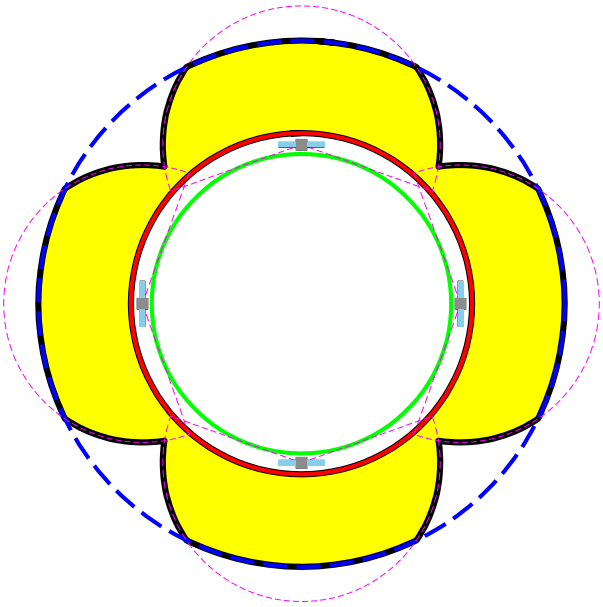


(b) Constraint Function (Single Coverage Area)

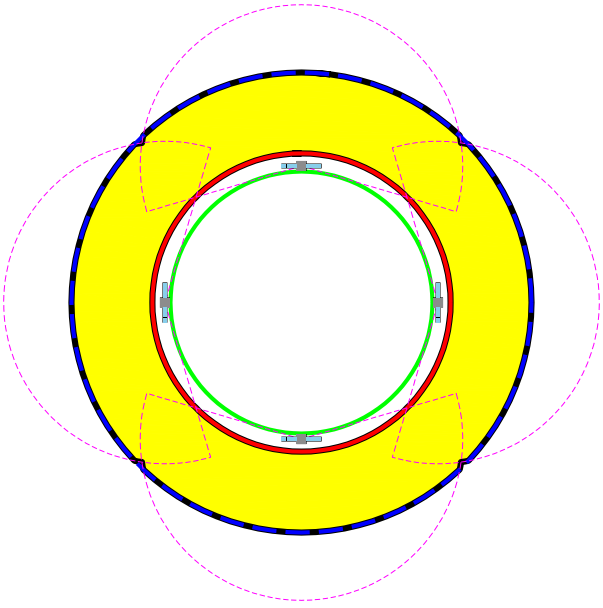


(c) Objective Function (Financial Cost)

Figure 4.10: Example 2b – Constraint and Objective Contours (Case 2)



(a) Initial Guess



(b) Converged Solution

Figure 4.11: Example 2b – Initial Guess and Converged Solution (Case 2)

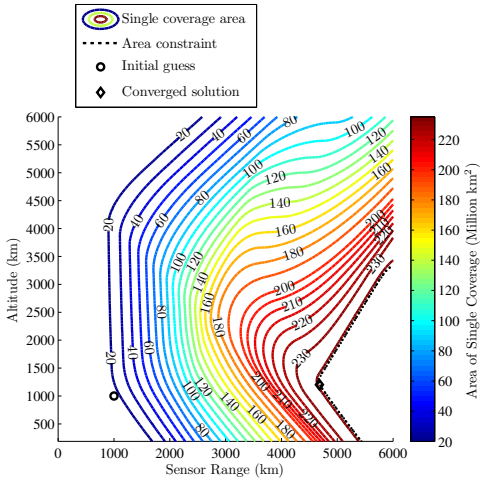
4.4.3 Example 2c – 10 Satellites

Finally, the largest, and most costly constellation analyzed in this example, the 10 satellite case also features (not surprisingly) the shortest necessary sensor range.

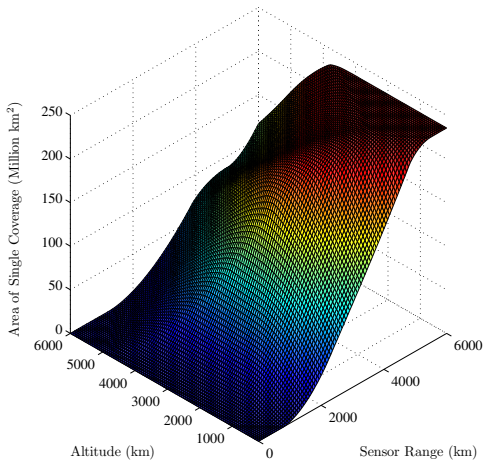
As with the previous cases in this example, the optimality of the solution is clear in Figure 4.12, where it corresponds to a constrained minimum in deployment cost. Further, as with the previous examples, it is clear from Figure 4.13b that the sensor range corresponds to an intersection between neighboring range shells with the upper target altitude shell (UTAS).

Table 4.9: Example 2c – Parameters and Solution

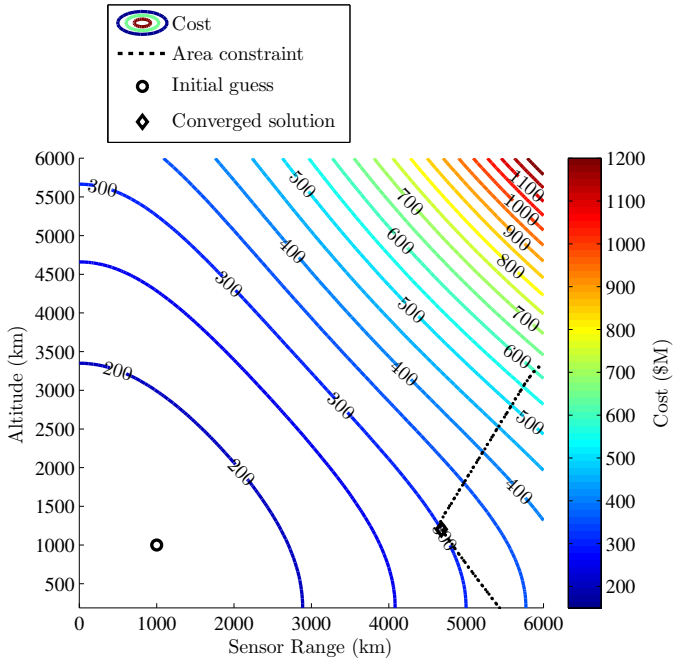
Parameter	Value	Description
n	10	number of satellites
$A_{\min}$	$0.999A_{AS}$	area constraint
R_0	1000 km	initial guess, R
h_0	1000 km	initial guess, h
R_{opt}	4680.102 km	converged cost-optimal R
h_{opt}	1202.932 km	converged cost-optimal h
Γ_{opt}	300.108 \$M	converged cost at solution
$A_{1\times_{\text{opt}}} - A_{\min}$	1.461×10^{-2} km^2	coverage area surplus



(a) Constraint Function (Single Coverage Area)

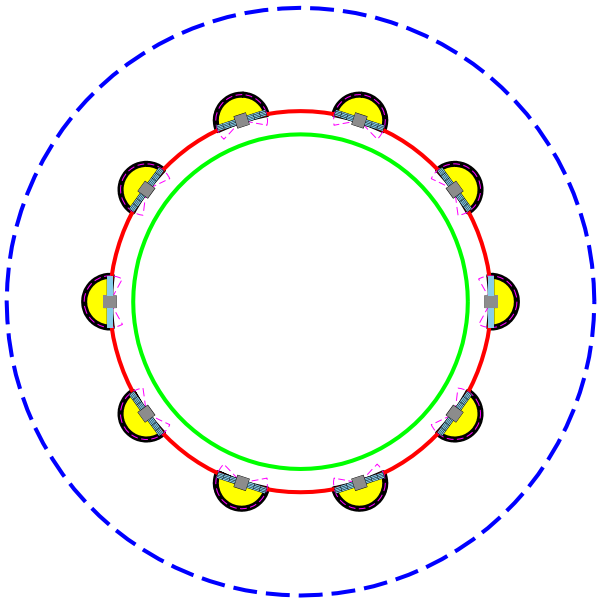


(b) Constraint Function (Single Coverage Area)

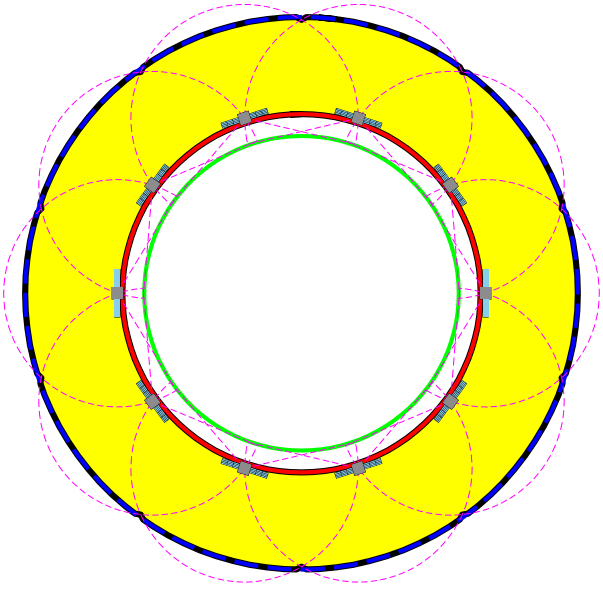


(c) Objective Function (Financial Cost)

Figure 4.12: Example 2c – Constraint and Objective Contours



(a) Initial Guess



(b) Converged Solution

Figure 4.13: Example 2c – Initial Guess and Converged Solution

4.4.4 Cost vs. Constellation Size

As mentioned in Section 4.4.2, the more well-behaved 4 satellite solution corresponds to the most cost-efficient constellation, while the 10 satellite solution is the worst (out of constellations analyzed). This result is, of course, relative to the exceedingly simple financial model presented in Section 4.1. By analyzing the remaining constellation sizes between 3 and 10 satellites (5 through 9 satellites), a trend is identified, as shown in Figure 4.14. This result clearly demonstrates the 4 satellite constellation's superiority in terms of cost efficiency within the context of this example. Solutions for the 5 through 9 satellite constellations are presented in Appendix B.2.

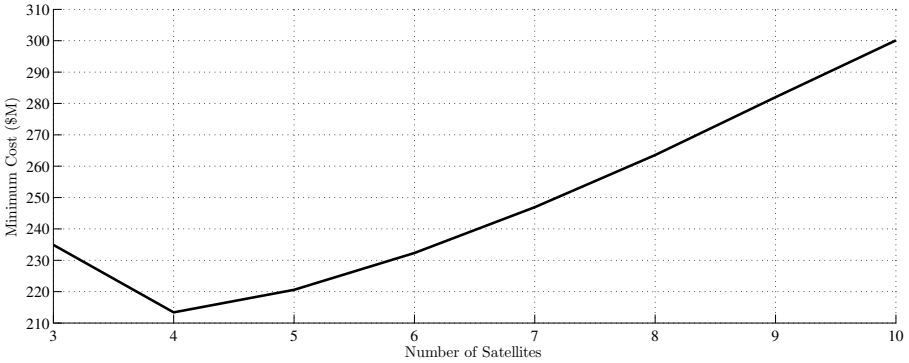


Figure 4.14: Example 2 – Optimal Deployment Cost for Constellations Providing Near-Total Single Coverage

4.5 Example 3 – Coverage as both an Objective and a Constraint

Double coverage area is maximized subject to two constraints – single coverage area must be at least 99.9% of the total area enclosed (as in Example 2), and deployment cost must not exceed a specified budget. The parameters describing the tangent height shell (THS) and dual-altitude band shell, AS, are shown in Table 4.10 along with the computed area inside AS.

Table 4.10: Example 3 – Parameters

Parameter	Value	Description
R_E	6378.14 km	assumed Earth radius
h_t	100 km	tangent height altitude
h_l	1000 km	lower target altitude
h_u	5000 km	upper target altitude
m	100 PPC	initial polygon resolution
A_{AS}	235,307,769 km^2	total area in AS
n	18	number of satellites
$A_{1\times_{\min}}$	$0.999A_{AS}$	area constraint
$\Gamma_{\max}$	515 \$M	deployment cost constraint
R_0	1000 km	initial guess, R
h_0	1000 km	initial guess, h

As with the preceding examples, satellites are assumed to be equally distributed in a single circular orbit, and are equipped with omni-directional sensors. Under these assumptions the satellite sensor range, R, and the satellite altitude, h, are sufficient to fully define a unique state of the system, resulting in a two-dimensional vector of optimization variables denoted by

$$\mathbf{x}_p = \begin{bmatrix} R & h \end{bmatrix}. \tag{4.17}$$

The performance index that is minimized with respect to $\mathbf{x}_p$ (double coverage

area) is given by

$$J = -A_{2\times}(R, h, n), \tag{4.18}$$

with the negative sign included with the understanding that minimization of $-A_{2\times}$ is equivalent to maximization of $A_{2\times}$.

Four inequality constraints are considered:

$$\mathbf{c} = \begin{bmatrix} A_{\min} - A_{1\times} \\ \Gamma(R, h, n) - \Gamma_{\max} \\ -R \\ h_{\text{ref}} - h \end{bmatrix} \leq \mathbf{0}, \tag{4.19}$$

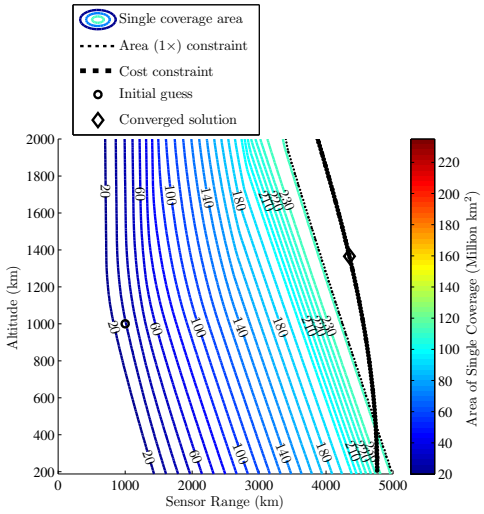
where the inequality is understood to apply element-wise between vectors $\mathbf{c}$ and $\mathbf{0}$. The constraint on deployment cost is necessary to ensure that there is a unique solution – without this budget, 100% single and 100% double coverage can be achieved with an infinite number of solutions of suitably large R and h values.

The converged solution, found in 16 SQP iterations, is shown in Table 4.11. Contours of the phase space are shown in Figure 4.15. The area and cost constraints are drawn as light and heavy dotted lines respectively. Recall that the area constraint corresponds to a minimum acceptable single coverage area, i.e. the solution is bounded to the left by the area constraint in Figure 4.15. Similarly, the cost constraint dictates the maximum possible deployment cost, i.e. the solution is bounded to the right by the cost constraint. Thus, the solution must (and does), exist between these two boundaries. A solution would not exist for a significantly reduced budget, as the single coverage area constraint would then be impossible to satisfy (the small feasible region between the constraints would not exist).

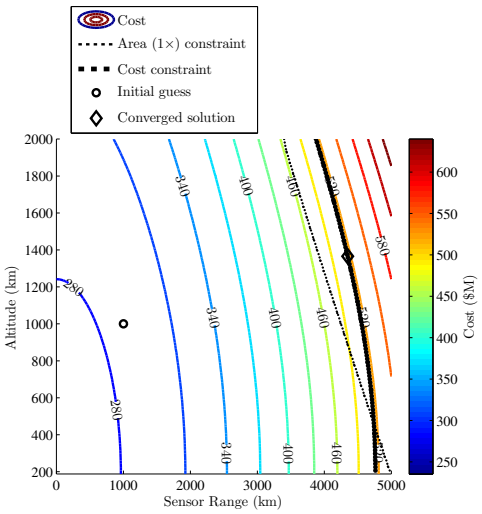
Figure 4.16 shows the initial guess and converged solution coverage configurations. Note that 100% single coverage is achieved in the double coverage-optimal solution.

Table 4.11: Example 3 – Solution

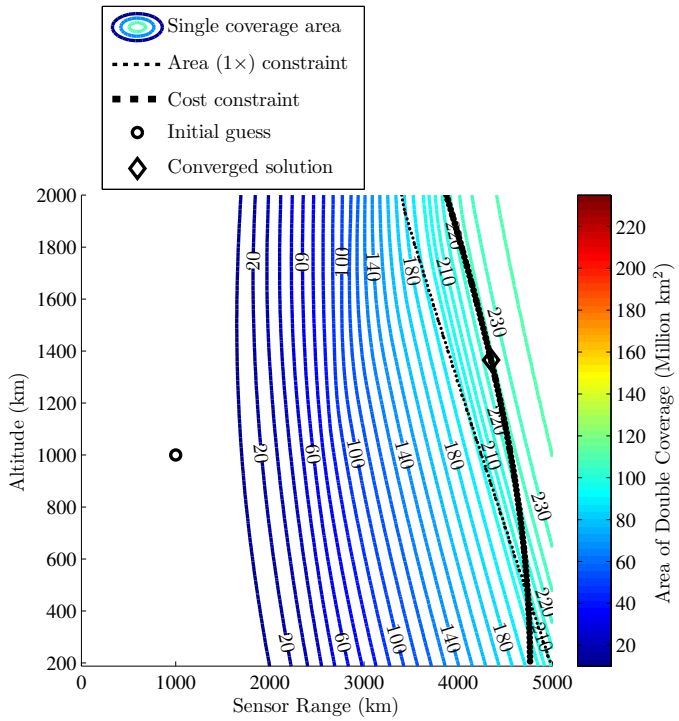
Parameter	Value	Description
R_{opt}	4348.122 km	converged area(2×)-optimal R
h_{opt}	1365.850 km	converged area(2×)-optimal h
$A_{2\times_{\text{opt}}}$	224,057,561 km^2	double coverage area at solution
$\Gamma_{\text{opt}} - \Gamma_{\text{max}}$	1.023×10^{-12} \$M	budget overrun
$A_{1\times_{\text{opt}}} - A_{\text{min}}$	2.353×10^5 km^2	single coverage area surplus



(a) Constraint Function (Single Coverage Area)

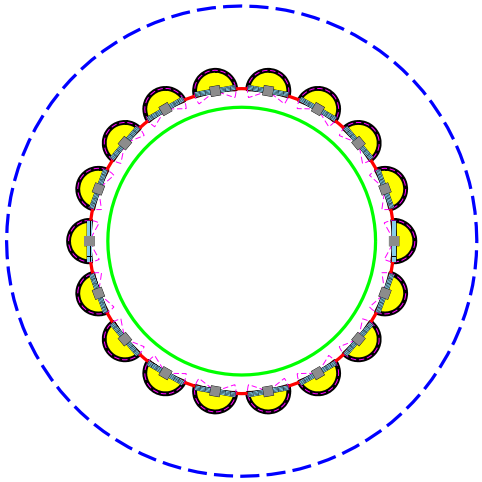


(b) Constraint Function (Financial Cost)

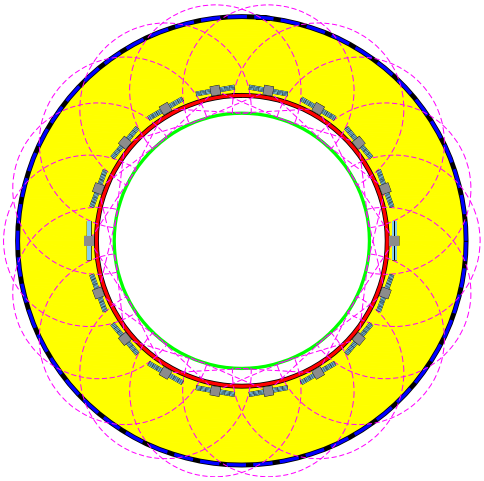


(c) Objective Function (Double Coverage Area)

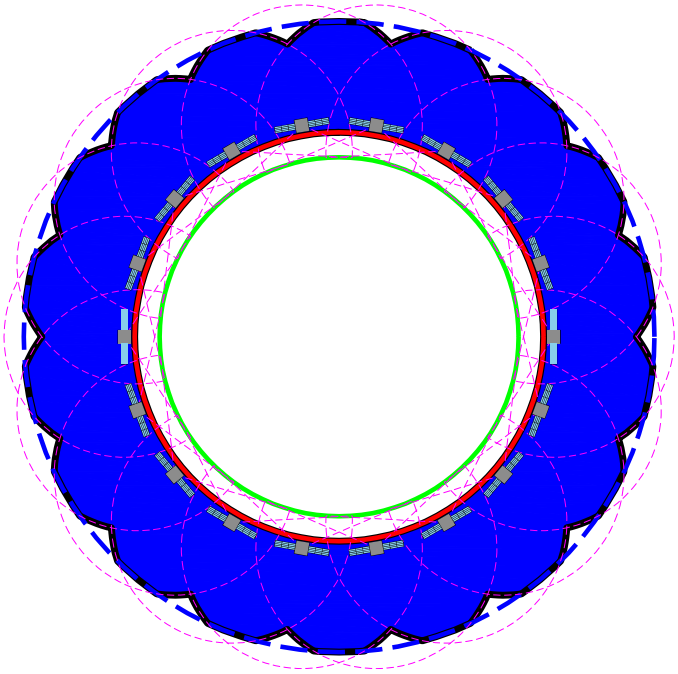
Figure 4.15: Example 3 – Constraint and Objective Contours



(a) Initial Guess



(b) Single Coverage at Converged Solution



(c) Double Coverage at Converged Solution

Figure 4.16: Example 3 – Constraint and Objective Contours

4.6 Example 4 – Arbitrary Sensor Profile

This problem illustrates how arbitrary sensor profiles can be used with the numerical ATH coverage models developed in Section 3.2 to design constellations. The satellite sensor cross-section considered here is by no means based on any real-world hardware. Prior to addressing the actual design problem, a brief discussion is presented that illustrates how the effective satellite range shell, RS_E is obtained, starting from a simple hand-drawn sketch.

4.6.1 Defining the Arbitrary Effective Range Shell

One of the fundamental advantages of the numerical analysis of ATH coverage is that investigations are not restricted simply to the omni-directional sensor case, or any case where an analytical geometric representation of the sensor cross-section can be derived. Consider the case shown in Figure 4.17. This sketch represents half of an in-plane sensor cross-section, and was hand-drawn by the investigator. There are no exact mathematical equations governing the path, making it 'arbitrary' for the purposes of this discussion.

Upon importation of the drawing via digital scanner, it is slightly adjusted using a graphics program to smooth the boundary and correct the angle at the end of the curve, making it perpendicular to the x direction (as intended). The resulting image is then analyzed using a figure analysis software package,[31] producing a series of (x, y) coordinates describing the half-curve. In MATLAB, the half-curve is imported, the sharp tip is moved to the origin, and the lower half is generated by replicating the (x, y) coordinates in reverse, while changing the y coordinates to

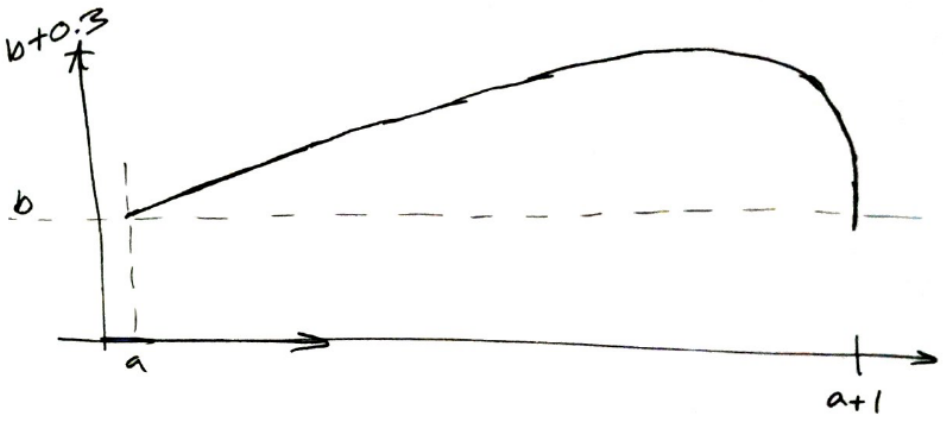


Figure 4.17: Example 4 – A 'Back of the Envelope' Arbitrary Sensor Cross-Section

their negative reciprocals. The coordinates are then scaled to produce a teardrop-shaped sensor cross-section that is of unit length, with a maximum width of 0.6. The sharp tip is centered on the spacecraft, as shown in Figure 4.18. This curve can then be scaled to any length, R, simply by multiplying all (x, y) coordinates by R (prior to any coordinate translation, of course).

Further, an in-plane attitude of the spacecraft (which is now a parameter, as the sensor is no longer omni-directional), or flight path angle, α can be defined. This angle is defined relative to the plane that is orthogonal to the satellite position vector. Traditionally, this plane is referred to as the local horizon. However, in this study, 'horizon' refers to a circular boundary around the Earth, as discussed in Chapter 1, and the two concepts should not be confused. Flight path angle is shown in Figure 4.19. Due to the symmetry about the satellite look-axis, denoted by the

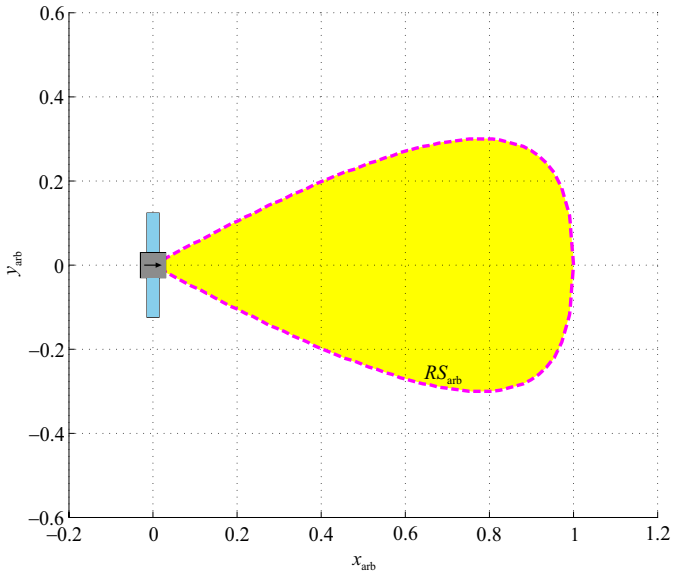


Figure 4.18: Example 4 – Arbitrary Sensor Cross-Section Imported to MATLAB

arrow on the satellite body in Figure 4.19, flight path angle is constrained by

$$-\frac{\pi}{2} \leq \alpha \leq \frac{\pi}{2}. \tag{4.20}$$

Once scaled, translated, and rotated as necessary, this arbitrarily shaped cross-section represents the sensor range shell, RS_{arb}.

Although this arbitrary sensor cross-section represents a tear-drop shape, *any* shape could have been used, and placed anywhere in relation to the space-craft. The only caveat is that the sensor region must be assumed to possess some form of symmetry across the orbit-plane such that ATH coverage measured in-plane somehow corresponds to ATH coverage in a three-dimensional volumetric sense.

Having defined the arbitrary range shell, RS_{arb}, the region corresponding to the tangent height triangle (THT) is removed in order to form the arbitrary *effective*

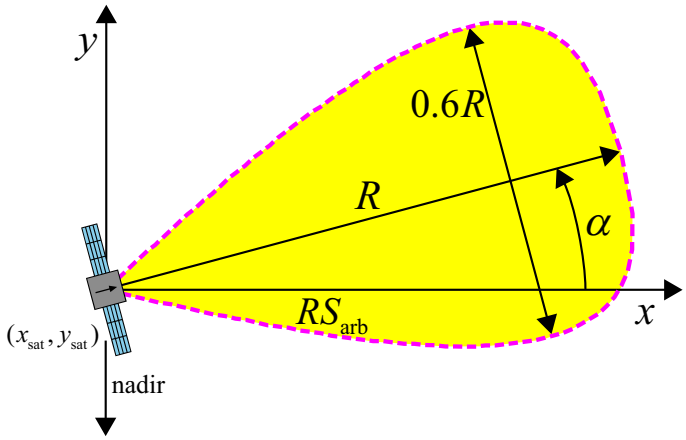


Figure 4.19: Example 4 – Arbitrary Sensor Cross-Section Notation

range shell, RS_E. As discussed in Section 3.2.1.2, this involves the formation of a polygon representing the THT, that is used to perform the polygon clipping operation $RS - \text{THT} = RS_E$. The two far-side vertices of the THT can be determined by simple geometry, and the third vertex is coincident with the satellite. The clipping operation is illustrated in Figure 4.20, where the THT is represented by the dashed triangle, and the shaded region represents the effective range shell, RS_E. The extent of the THT need only exceed the maximum extent of RS_{arb} to ensure there are no leftover regions beyond the THT after the clipping operation. In this example, the distance to the far side of the THT (along the look-axis) is chosen to be $1.1R$.

Possessing the means to generate the arbitrary effective range shells, it is then straightforward to utilize the numerical ATH coverage models developed in Section 3.2 to evaluate the coverage they provide.

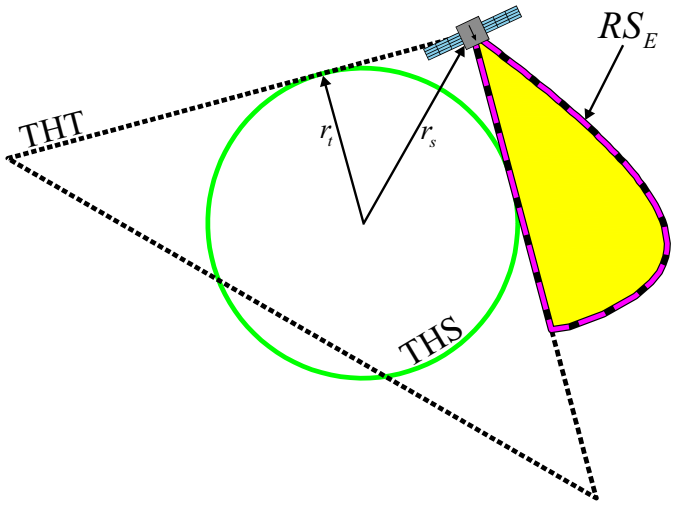


Figure 4.20: Example 4 – Clipping the THT From RS_{arb} to Form RS_E

4.6.2 Problem Statement and Solution

The problem solved here can be considered an extension of the problem addressed in Example 1 – maximization of single ATH coverage subject to a deployment constraint. First, it is important to note that the financial model parameters are slightly modified for this example, as shown in Table 4.1, where b_{arb} is used. This parameter is adjusted from b_{OD} to reflect the reduced coverage area relative to the sensor range, R.

First, the parameter vector is formed, just as in Example 1, but with the addition of the flight path angle, α (uniform across the constellation):

$$\mathbf{x}_p = [R \quad h \quad \alpha]. \tag{4.21}$$

Next, the objective function is defined as

$$J = -A_{1\times}(R, h, \alpha, n), \tag{4.22}$$

which is to be minimized (to maximize single coverage). This problem is subject to five inequality constraints that are written in vector form as

$$\mathbf{c} = \begin{bmatrix} \Gamma(R,h,n) - \Gamma_{\max} \\ -R \\ h_{\text{ref}} - h \\ \alpha - \frac{\pi}{2} \\ \frac{\pi}{2} - \alpha \end{bmatrix} \leq \mathbf{0}. \tag{4.23}$$

Note that the financial model, described in Section 4.1, does not depend upon flight path angle, α.

The parameters used in this problem are shown in Table 4.12. The initial guess is intentionally chosen to be far away in all three dimensions from the expected (by intuition) solution. Also note that although 100 PPC is the polygon resolution used for the altitude shell, the arbitrary range shell as generated in MATLAB consists of 110 vertices. This is purely an artifact of the curve generation process, and an order-of-magnitude similarity with m is considered acceptable.

Table 4.12: Example 4 – Parameters

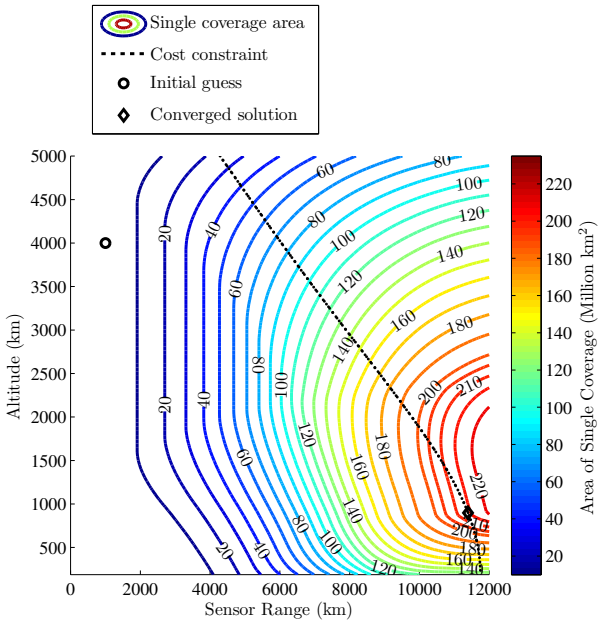
Parameter	Value	Description
R_E	6378.14 km	assumed Earth radius
h_t	100 km	tangent height altitude
h_l	1000 km	lower target altitude
h_u	5000 km	upper target altitude
m	100 PPC	initial polygon resolution
n	7	number of satellites
$\Gamma_{\max}$	250 \$M	deployment cost constraint
R_0	1000 km	initial guess, R
h_0	4000 km	initial guess, h
α_0	45°	initial guess, α

After 41 SQP iterations, the optimizer converges to the solution shown in Table 4.13. In order to visualize this three degree-of-freedom problem, contours at the solution are produced holding one variable fixed, creating a pair of contour plots for ATH coverage area and deployment cost. Figure 4.21 depicts overall and detail contours when holding flight path angle constant at the solution. Note that the detail contour (Figure 4.21b) uses a rescaled color map, and the colors do not correspond to the colors in the overall contour (Figure 4.21a). These figures illustrate that coverage area only decreases in any feasible direction in R or h (i.e. not across the constraint boundary). Similarly, holding h fixed at the solution yields the contours shown in Figure 4.22. Just as in the case of constant flight path angle, any variation in R and α in a feasible direction results in a decrease in ATH coverage area. As with earlier examples constrained by deployment cost, because of the quadratic behavior of the cost function in both R and h, the solution must fall on or to the left of the constraint boundary.

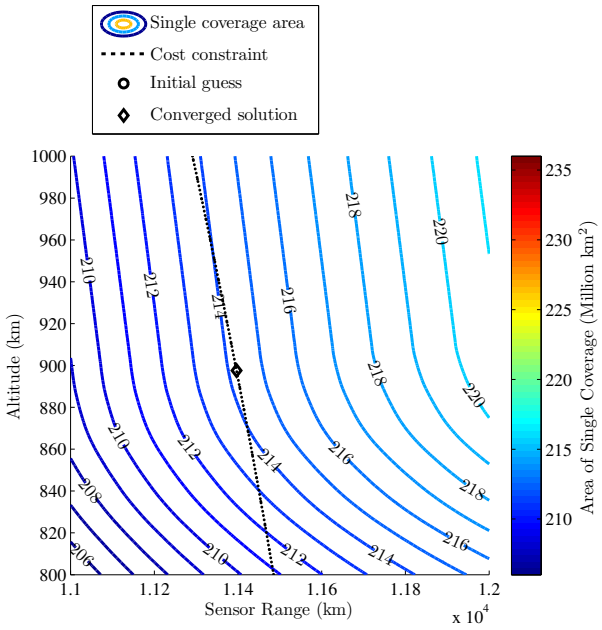
While these contour illustrations are by no means sufficient condition to declare optimality of the solution, the SQP optimizer in *fmincon* does return a flag indicating that it has found a constrained locally optimal solution. The MATLAB SQP implementation utilizes a variation of the Karush-Kuhn-Tucker (KKT) conditions[32] as one of several criteria to determine optimality. Generally, KKT is only a necessary condition for optimality. However, for continuously differentiable and convex objective and constraint functions (as is the case in this problem), KKT also provides sufficient condition for optimality.[33]

Table 4.13: Example 4 – Solution

Parameter	Value	Description
R_{opt}	11396.402 km	converged area-optimal R
h_{opt}	897.559 km	converged area-optimal h
α_{opt}	-5.937°	converged area-optimal α
$A_{1\times_{\text{opt}}}$	214,239,724 km^2	single coverage area at solution
$\Gamma_{\text{opt}} - \Gamma_{\max}$	0 \$M	budget overrun

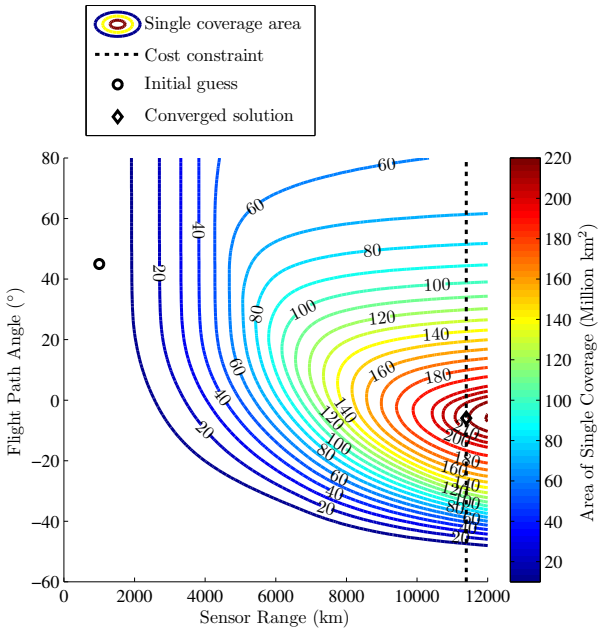


(a) Overall

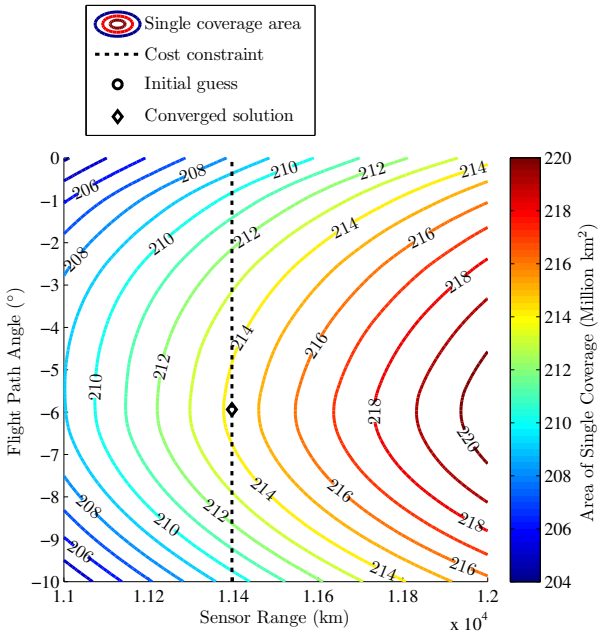


(b) Detail Near Solution

Figure 4.21: Example 4 – Objective Function (Single Coverage Area, Constant $\alpha = -5.937°$ at Solution)

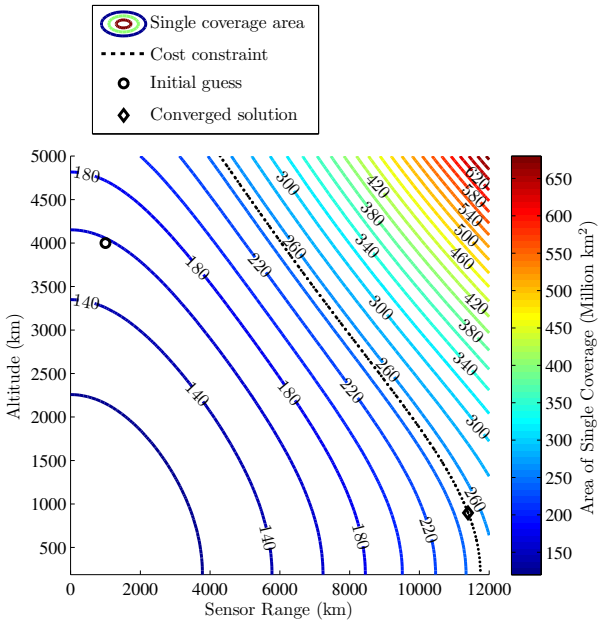


(a) Overall

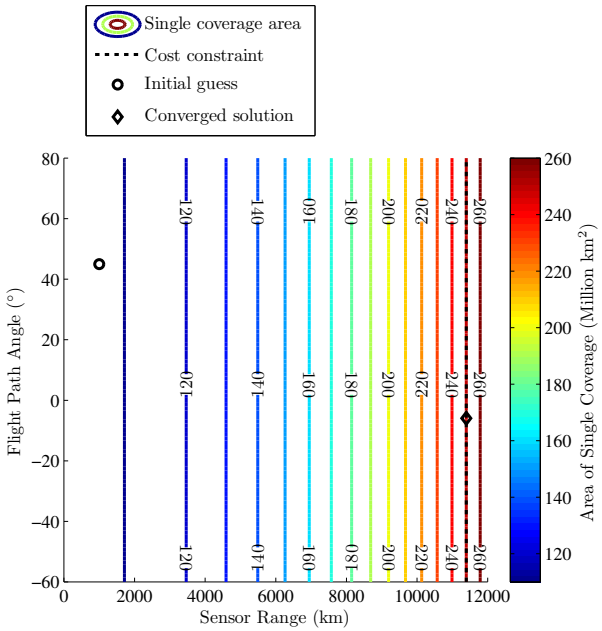


(b) Detail Near Solution

Figure 4.22: Example 4 – Objective Function (Single Coverage Area, Constant $h = 897.559$ km at Solution)

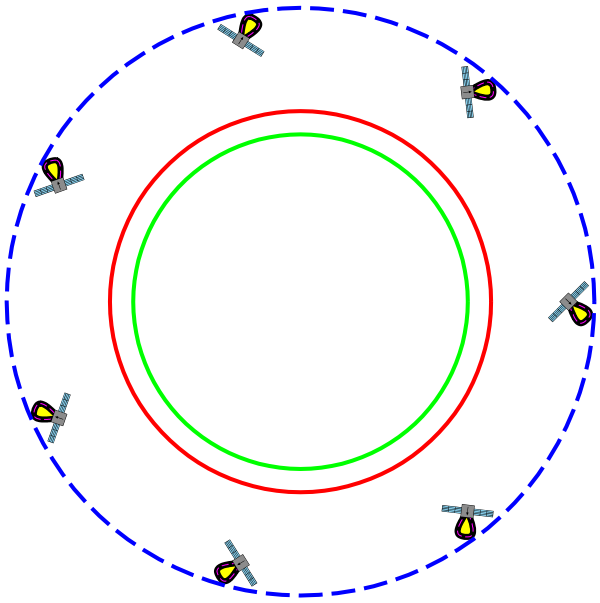


(a) Constant $\alpha = -5.937°$ at Solution

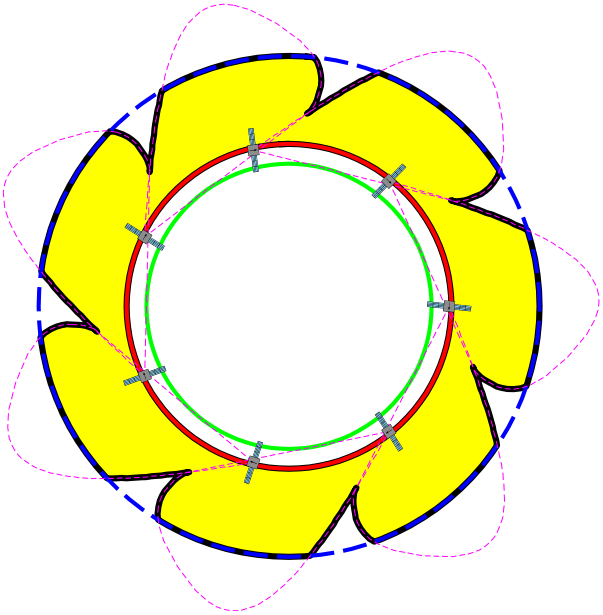


(b) Constant $h = 897.559$ km at Solution

Figure 4.23: Example 4 – Constraint Function (Deployment Cost)



(a) Initial Guess



(b) Converged Solution

Figure 4.24: Example 4 – Initial Guess and Converged Solution

Chapter 5

Conclusions

In this study numerical ATH coverage algorithms are developed and presented that allow computation of ATH coverage area for any desired coverage multiplicity. The approach described involves discretization (into polygons) of in-plane satellite coverage regions and regions of interest (assumed in this research to be a dual-altitude band shell bounded above and below by prescribed altitudes). The interactions between these polygons are analyzed by performing different sequences of polygon clipping operations, yielding a result polygon that represents the region exhibiting the desired coverage multiplicity bounded by the region of interest. When the enclosed area of this result polygon is computed, it provides an effective measure of in-plane ATH coverage for a given constellation configuration.

The necessary sequences of polygon clipping operations for this analysis are first developed using set notation with a focus on efficiency by avoiding unnecessary polygon clipping operations. These efficient ATH coverage algorithms are then implemented using several different polygon clipping implementations which are compared by their performance. The error introduced by approximating curvilinear regions with finite-resolution polygons is extensively analyzed, and relationships guiding appropriate polygon resolution selection for a desired accuracy are developed.

Several example problems are then solved that illustrate how the numerical ATH coverage algorithms can be incorporated into a satellite constellation design problem. Cases using both omni-directional and arbitrary sensor profiles are considered. Using a parameter optimization code, the coverage algorithms are first used as objective functions to maximize ATH coverage, then as constraints requiring a target ATH coverage amount be achieved. Simple though they may be, these examples demonstrate the utility of the numerical ATH coverage algorithms in the design of constellation configurations for ATH coverage.

It is worth emphasizing the generality this methodology allows in the analysis of ATH coverage. First, the sensor cross-sections can be of any shape, and can even be unique to each satellite in a constellation. Secondly, the analysis can be used in both time-invariant and time-varying analyses – the only two time-varying parameters necessary are the locations and in-plane attitudes of each spacecraft at a given time. Lastly, although the current study only considers the dual-altitude band shell as the region of interest, it is possible to analyze for coverage in regions of interest that are of any desired planar form. For instance, the region of interest could be fixed above a certain location on the central body, i.e. to analyze for ATH coverage specifically above a prescribed geographic region.

5.1 Future Work

The fundamental limitation of the research presented here is that all satellites are assumed to remain in a single plane, and their sensor profiles are assumed to exhibit some form of symmetry across the orbit-plane. This symmetry is nec-

essary so that in-plane coverage can be assumed as representative of volumetric coverage in a three-dimensional sense. Expanding to the three-dimensional problem explicitly, one possible approach is to analyze the problem as interactions between various polyhedra representing the coverage regions and regions of interest. However, the existing numerical methods for the analysis of the intersection between three-dimensional polyhedra are far less general than in the two-dimensional case investigated in this study, and typically require that the polyhedra be convex. Such an approach requires convex decomposition of non-convex polyhedra in order to perform the necessary operations between them.

Alternatively, a future extension to this research to address the volumetric case (requiring additional computational resources) could be to analyze in-plane ATH coverage within a series of level planes offset perpendicularly from the orbital plane. Using this technique, volumetric coverage can be approximated layer-by-layer, much in the same fashion that some three-dimensional printers generate prototype objects. This approach could be implemented using the ATH coverage algorithms developed in this study and would only require some method to determine offset-plane cross-sections of the sensor regions and regions of interest.

Appendices

Appendix A

Rederivation of the Analytical ATH Coverage Model

This appendix presents a rederivation of the analytical above the horizon (ATH) coverage model developed by Marchand and Kobel.[4] The motivations for carrying this out are two-fold: first, it proves to be an invaluable exercise to develop a familiarity with the overall problem. Secondly, the resulting analytical implementation is instrumental in verifying the numerical models developed for the current research problem.

The resulting piecewise-differentiable set of functions developed here differs slightly in form from that found by the original investigators,[4] but is geometrically equivalent and produces the same results.

A.1 Introduction

Marchand and Kobel[4] present a coverage model that, given geometric parameters of the Earth, regions of interest (dual-altitude band shell), and satellite position, and sensor capability, returns the ATH coverage area (in a planar analysis). This coverage can then be used as part of a design process to determine sets of parameters providing the desired amount of ATH coverage. This analysis is subject to a number of simplifying assumptions. First, the three-dimensional scenario is reduced to a planar case. This results in no loss of generality when also assum-

ing omni-directional sensor profiles for the satellite (i.e. spherical, centered on the satellite). This assumption produces circular in-plane sensor profile cross-sections. Additionally, only circular satellite orbits are considered – this assumption makes the amount of coverage time-invariant. Finally, only the coverage provided by a single satellite is considered.

The result is a piecewise-smooth analytical model that explicitly provides the coverage area (which correlates to coverage volume, in the three-dimensional case) for a given set of parameters in the dual-altitude band ATH coverage problem. As is shown in Section 3.1.6, this analytical model allows very rapid calculation of coverage area relative to its numerical counterparts. Numerical methods allow analysis of a much more generalized set of problems at a cost of performance and accuracy. However, the analytical results are fundamentally important to the research presented in this thesis for validation as well as development of intuition by the investigator.

A.2 Problem Geometry and Notation

A typical problem configuration is shown in Figure A.1. The state can be completely defined by five fundamental quantities: The radii of the 4 drawn circles (described in the proceeding sections), and the distance between S and E, which denote the locations of the satellite and center of the Earth respectively.

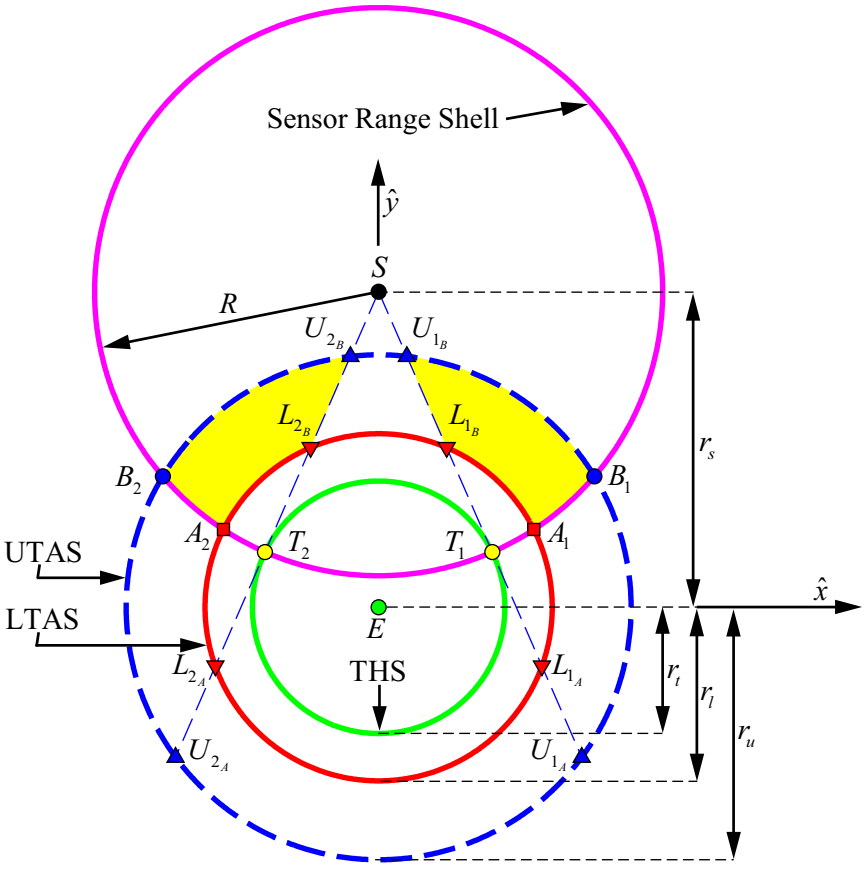


Figure A.1: Problem Notation

A.2.1 Tangent Height Shell – THS – Radius r_t

The innermost circle centered on the Earth represents the tangent height shell (THS). This shell, with radius r_t, determines the below the horizon (BTH) region in the direction of the Earth. From a practical standpoint, one can imagine this as representing the radius of the outer reaches of the Earth's atmosphere, reflecting light or causing other disturbances that negate the function of the satellite's on-board sensors.

Practical considerations require that r_t be greater than or equal to the radius of the Earth's surface.

A.2.2 Lower Target Altitude Shell – LTAS – Radius r_l

The lower target altitude shell (LTAS), with radius r_l, centered on the Earth, specifies the lower bound of the altitude band of interest. Any ATH coverage area resulting below this altitude is of no interest to the investigation.

The LTAS must be larger than the THS and smaller than the UTAS.

A.2.3 Upper Target Altitude Shell – UTAS – Radius r_u

The upper target altitude shell (UTAS), with radius r_u, centered on the Earth, denotes the upper altitude bound. The LTAS and UTAS define an annulus-shaped region of interest, where the investigator seeks to determine the amount sensor coverage provided by the satellite. The only constraint on the size of the UTAS is that it must be larger than the LTAS.

A.2.4 Sensor Range Shell – RS – Radius R

The sensor range shell (RS), with radius R, centered on the satellite, denotes the range for an omni-directional satellite sensor. The sensor range can be any non-negative value.

A.2.5 Circular Satellite Orbit – Radius r_s

Under the previously discussed assumption that the satellite in question has a circular orbit, its altitude is constant, and thus the radius of its orbit, r_s, is also constant. In the context of this problem, r_s can be taken to be the separation between the center of the Sensor Range Shell and the centers of any of the other shells (that are of course concentric with one another).

A.2.6 Tangent Height Cone – THC

The in-plane cross-section of the tangent height cone (THC) is shown in Figure A.1 as the central triangular region bounded by the straight dashed lines between points S, U_{1_A}, and U_{2_A} (although it actually extends to infinity, away from S). While the THC is not explicitly described as the aforementioned parameters, r_t, r_l, r_u, r_s, and R are, it is a useful concept in the development of the analytical ATH coverage model. The THC geometry depends upon the satellite orbit radius, and the radius of the THS. As shown in the figure, the THC represents the BTH region in the direction of the Earth.

A.2.7 Shell and Line Intersection Points

The various intersections between the sides of the THC and the 4 fundamental shells are shown in Figure A.1. Their nomenclature is consistent with the work published by Marchand and Kobel,[4] in order to simplify comparison between the two results.

Intersection points calculated in this analysis are determined identically to the methods clearly outlined by Marchand and Kobel,[4] and are thus not reproduced here. Suffice it to say, the locations of the various intersections of interest are determined by simple geometry and solving the equations of various pairs of circles and lines simultaneously.

A.3 Fundamental Area Elements

It is desired to determine the area enclosed in the various complex regions listed in the described piecewise-smooth ATH coverage model in Section A.4. While the coverage regions themselves are generally complicated in shape, they are reduced to the simple addition and subtraction of the areas of various fundamental regions that can be computed by geometry. This section outlines the different elements, their slightly modified notation (relative to the published results by Marchand and Kobel),[4] and diagrams to denote the variables referring to each dimension. Note that the composite triangle of the first kind, A_{Λ_1}, used by Marchand and Kobel is not used in the current formulation, and is thus omitted from this section. However, the composite triangle of the second kind A_{Λ_2} is used, and the 2 subscript is maintained to avoid introducing additional confusion when comparing the two results.

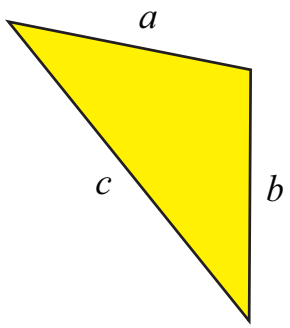


Figure A.2: Area of Triangular Region – $A_\Delta(a, b, c)$

All of the area elements described in this section depend upon quantities taken either directly from the problem parameters (r_t, r_l, r_u, r_s, R), or they are easily obtained after determination of the intersection points described in the literature.[4]

A.3.1 Triangular Regions – $A_\Delta(a, b, c)$

Given a potentially scalene triangle with three sides with lengths a, b, and c as shown in Figure A.2, the semi-perimeter is defined as

$$s \equiv \frac{a+b+c}{2}, \tag{A.1}$$

and thus, by Heron's Formula,[34] it can be shown that the area enclosed within the triangle is given by

$$A_\Delta(a, b, c) = \sqrt{s(s-a)(s-b)(s-c)}. \tag{A.2}$$

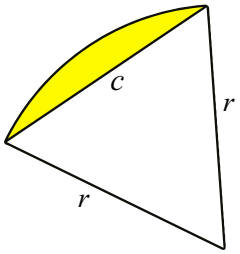


Figure A.3: Area of Circular Segment – $A_\Sigma(r, c)$

A.3.2 Circular Segments – $A_\Sigma(r, c)$

The circular segment shown in Figure A.3 is determined (as shown by Marchand and Kobel)[4] by the relation

$$A_\Sigma(r, c) = r^2 \sin^{-1}[c/2r] - (c/4)\sqrt{4r^2 - c^2}, \tag{A.3}$$

where r is the radius of the circle, and c is the chord length of the segment.

A.3.3 Circular Sectors – $A_{\pi_1}(r, c)$

The area of a simple sector of a circle with radius r, shown in Figure A.4, is found by $A = (1/2)r^2\lambda$, where λ is the interior angle (in radians) of the triangular portion, opposite the chord, c. Given c and r, λ is determined by trigonometry to be

$$\lambda = 2\sin^{-1}\left(\frac{c}{2r}\right), \tag{A.4}$$

which can then be used to compute the area;

$$A_{\pi_1}(r, c) = r^2 \sin^{-1}\left(\frac{c}{2r}\right). \tag{A.5}$$

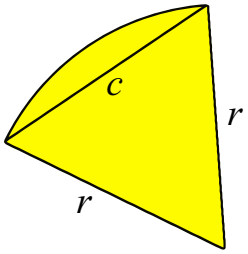


Figure A.4: Area of Circular Sector – $A_{\pi_1}(r, c)$

A.3.4 Composite Teardrop Sectors – $A_{\pi_2}(r, R, r_s)$

The area of the region shown in Figure A.5 is determined by the summation of the area of a triangle, A_Δ, and a circular segment, A_Σ, or it's complement (i.e. the rest of the area inside the circle, excluding the circular segment). In order to do this, the chord length c (distance between the intersection points of the two circles) is determined.

First, define a coordinate system where the y-axis lies along r_s, starting at the origin O and pointing in the direction of the satellite, S. The x-axis begins at the origin and points perpendicularly to r_s (to the right in Figure A.5), creating a standard right-handed coordinate system. Given values of radii for the range shell, the altitude shell, (R and r respectively), and the distance between their centers (r_s), their positions are

$$\begin{aligned} y_{int} &= \frac{r^2 + r_s^2 - R^2}{2r_s}, \\ x_{int} &= \pm\sqrt{r^2 - y_{\text{int}}^2}, \\ c &= 2|x_{\text{int}}|, \end{aligned} \tag{A.6}$$

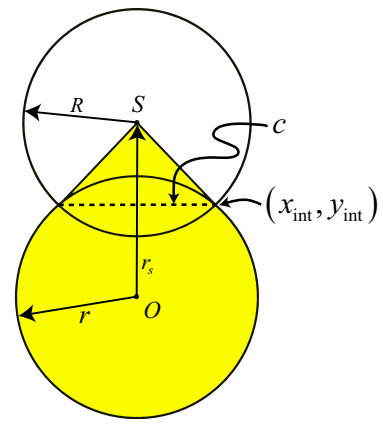


Figure A.5: Area of Composite Teardrop Sector – $A_{\pi_2}(r, R, r_s)$

which then make it possible to define

$$A_{\pi_2}(r, R, r_s) = \begin{cases} A_\Delta(r, c, r) + A_\Sigma(r, c) & : R > \sqrt{r_s^2 + r^2} \\ A_\Delta(r, c, r) + 2\pi r^2 - A_\Sigma(r, c) & : R \leq \sqrt{r_s^2 + r^2} \end{cases}, \tag{A.7}$$

with c defined as above.

A.3.5 Composite Triangle of the Second Kind – $A_{\Lambda_2}(r, a, b, c)$

The area of the region shown in Figure A.6 is determined by the difference between a triangle (A_Δ) and a circular segment (A_Σ). With this in mind, it is clear that

$$A_{\Lambda_2}(r, a, b, c) = A_\Delta(a, b, c) - A_\Sigma(r, b), \tag{A.8}$$

where a, b, and c are sides of the initial triangle, and r is the radius of curvature of the 'missing' circular segment on side b.

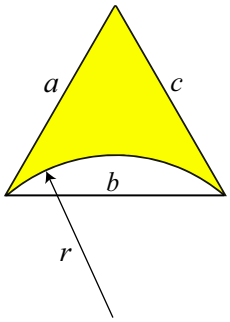


Figure A.6: Area of Composite Triangle of the Second Kind – $A_{\Lambda_2}(r, a, b, c)$

A.3.6 Intersection of Altitude and Range Shells – $A_{AS\cap RS}(r, R, r_s)$

The shaded area shown in Figure A.7 is determined as a function of the previously defined fundamental area elements. It can be shown that generalized determination of this area is broken down into four major cases.

Case 1a: $r > R + r_s$ or $R > r + r_s$

The first case, referred to as Case 1a, occurs when one shell is completely encompassed inside the other. This case is checked for first, because computation of two of the other cases requires determination of the intersection points between the two shells (that do not exist in this case). This condition can be easily detected, and the area computed as

$$A_{AS\cap RS}(r, R, r_s) = \begin{cases} \pi R^2 & : r > R + r_s \\ \pi r^2 & : R > r + r_s \end{cases} . \tag{A.9}$$

Case 1b: $r_s \geq R + r$

The trivial case, 1b, occurs when the range shell and altitude shell are sufficiently far apart that there is no overlap at all, thus the area is

$$A_{AS \cap RS}(r, R, r_s) = 0. \tag{A.10}$$

Case 2: $r > R$

Assuming the intersection points exist (because they did not in Case 1), the distance between the two intersection points of the range and altitude shells must be determined. This distance, c, can be readily calculated by Equation A.6. Case 2 occurs if the range shell has a larger radius, R, than the altitude shell radius, r. This case is then divided into two sub-cases,

$$A_{AS \cap RS}(r, R, r_s) = \begin{cases} A_\Sigma(R, c) + A_\Sigma(r, c) & : r_s > \sqrt{r^2 - R^2} \\ A_\Sigma(r, c) + \pi R^2 - A_\Sigma(R, c) & : r_s \leq \sqrt{r^2 - R^2} \end{cases} . \tag{A.11}$$

Case 3: $r \leq R$

Finally, Case 3 occurs when the altitude shell is larger than the range shell. As with Case 2, the distance between the two intersection points, c, is computed, as in Equation A.6. The area is then be computed as

$$A_{AS \cap RS}(r, R, r_s) = \begin{cases} A_\Sigma(R, c) + \pi r^2 - A_\Sigma(r, c) & : r_s \leq \sqrt{R^2 - r^2} \\ A_\Sigma(R, c) + A_\Sigma(r, c) & : r_s > \sqrt{R^2 - r^2} \end{cases} . \tag{A.12}$$

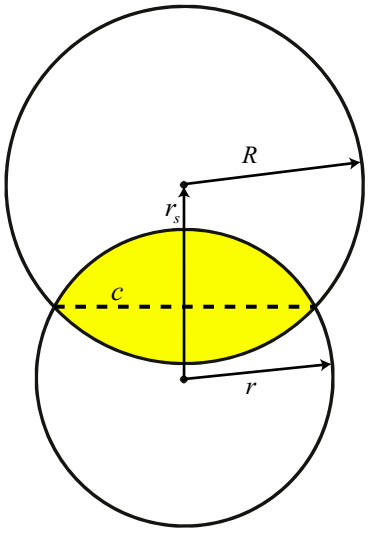


Figure A.7: Area of Intersection of Altitude and Range Shells – $A_{AS \cap RS}(r, R, r_s)$

A.4 Revised ATH Coverage Model

The coverage model presented in this section was constructed independently, but using the same fundamental area element definitions published by Marchand and Kobel.[4] The resulting equations were then compared to the published results for verification. In the context of this problem, the expressions governing the coverage model are not unique, thus the definitions presented here differ slightly from the published results. As a consequence of these different definitions, the original 18 sub-cases presented in the literature are reduced to the 12 presented here.

The now considered 12 sub-cases are divided into 3 different groups, depending upon whether $r_s < r_l$ (Case 1), $r_l \leq r_s < r_u$ (Case 2), or $r_u \leq r_s$ (Case 3). Each sub-case description below provides explicit conditions under which each

piecewise-smooth segment of the ATH coverage model applies.

A.4.1 Case 1 – $r_s < r_l$

A.4.1.1 Case 1a

Conditions:

$$\begin{aligned} r_s &< r_l \\ R &< |\overline{L_{1_A} S}| \end{aligned} \tag{A.13}$$

Area of coverage:

$$A = A_{AS \cap RS}(r_u, R, r_s) - A_{AS \cap RS}(r_l, R, r_s) \tag{A.14}$$

Diagram:

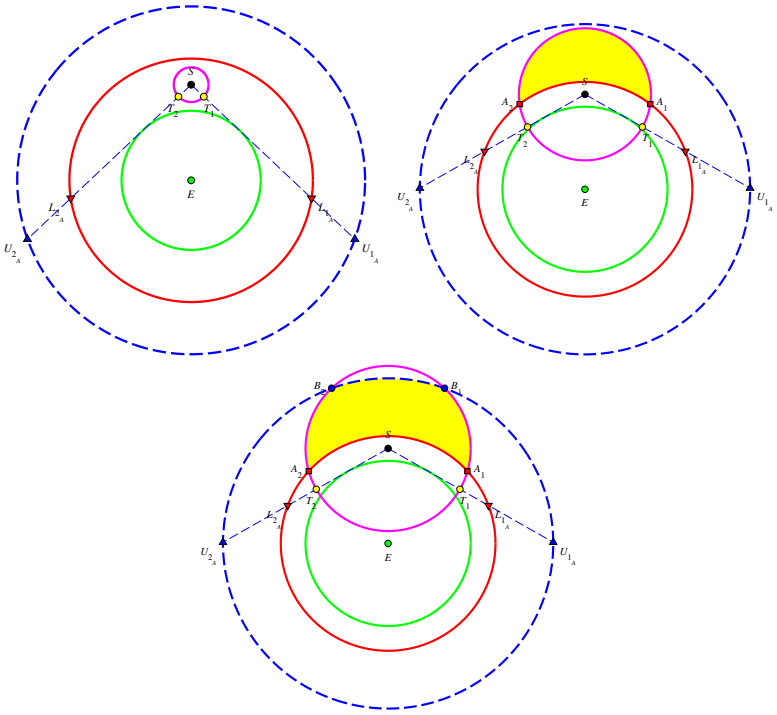


Figure A.8: Variations of Case 1a

A.4.1.2 Case 1b

Conditions:

$$\begin{gathered} r_s < r_l \\ |\overline{L_{1_A}S}| \leq R < |\overline{U_{1_A}S}| \end{gathered} \tag{A.15}$$

Area of coverage:

$$A = A_{AS\cap RS}(r_u, R, r_s) - A_{\pi_1}(R, |\overline{T_1T_2}|) - \pi r_l^2 + A_{\pi_2}(r_l, |\overline{L_{1_A}S}|, r_s) \tag{A.16}$$

Diagram:

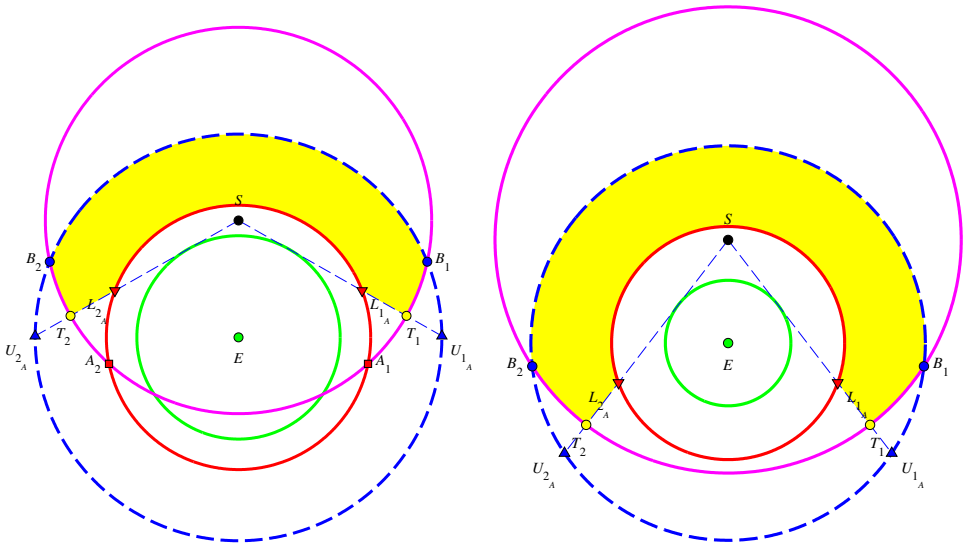


Figure A.9: Variations of Case 1b

A.4.1.3 Case 1c

Conditions:

$$\begin{aligned} r_s &< r_l \\ |\overline{U_{1_A}S}| &< R \end{aligned} \tag{A.17}$$

Area of coverage:

$$A = \pi r_u^2 - \pi r_l^2 - A_{\pi_2}(r_u, |\overline{U_{1_A}S}|, r_s) + A_{\pi_2}(r_l, |\overline{L_{1_A}S}|, r_s) \tag{A.18}$$

Diagram:

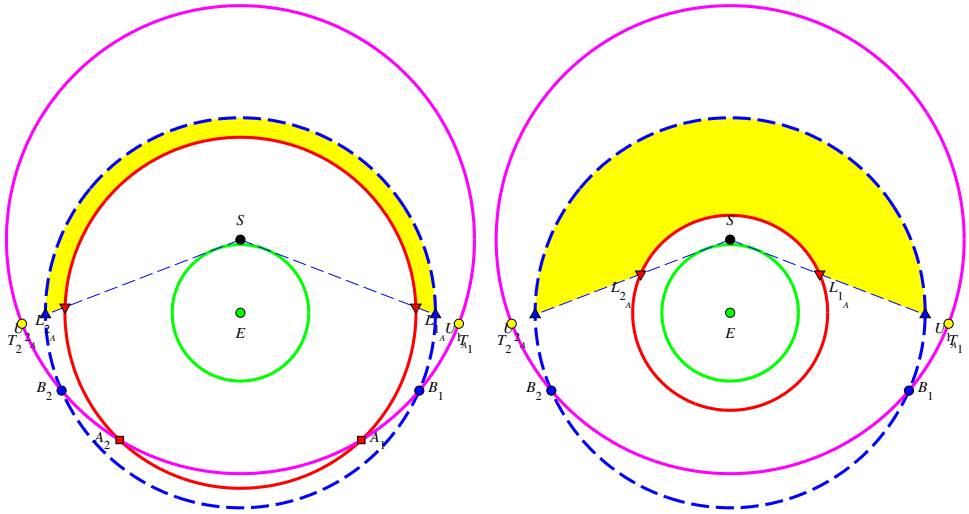


Figure A.10: Variations of Case 1c

A.4.2 Case 2 – $r_l \leq r_s < r_u$

A.4.2.1 Case 2a

Conditions:

$$\begin{aligned} r_l &\leq r_s < r_u \\ R &< |\overline{L_{1_B} S}| \end{aligned} \tag{A.19}$$

Area of coverage:

$$A = A_{AS \cap RS}(r_u, R, r_s) - A_{\pi_1}(R, |\overline{T_1 T_2}|) \tag{A.20}$$

Diagram:

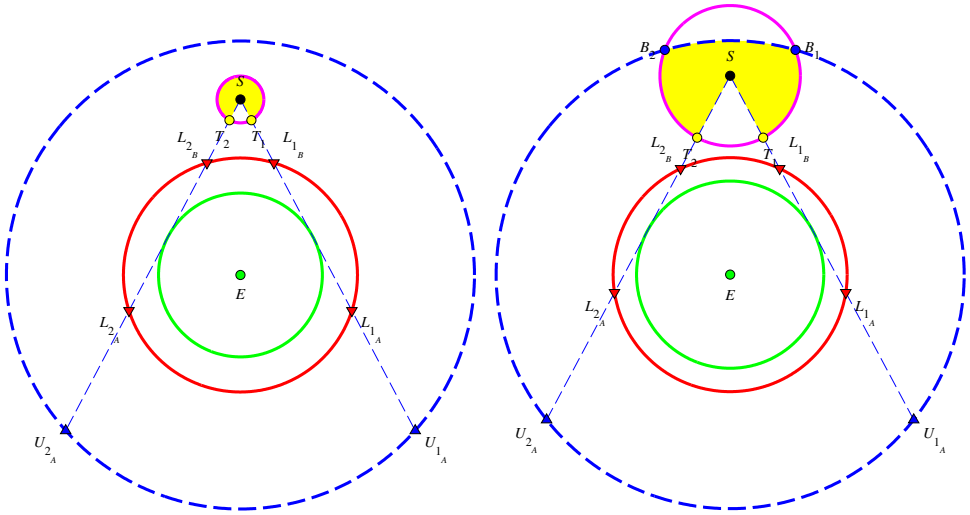


Figure A.11: Variations of Case 2a

A.4.2.2 Case 2b

Conditions:

$$\begin{aligned} r_l \leq r_s < r_u \\ |\overline{L_{1_B}S}| \leq R < |\overline{L_{1_A}S}| \end{aligned} \tag{A.21}$$

Area of coverage:

$$\begin{aligned} A = A_{AS\cap RS}(r_u, R, r_s) - A_{AS\cap RS}(r_l, R, r_s) \\ -A_{\Lambda_2}(r_l, |\overline{L_{2_B}S}|, |\overline{L_{1_B}L_{2_B}}|, |\overline{L_{1_B}S}|) \end{aligned} \tag{A.22}$$

Diagram:

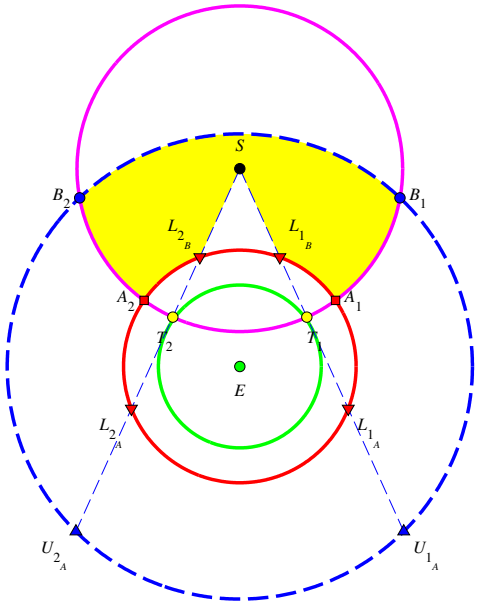


Figure A.12: Case 2b

A.4.2.3 Case 2c

Conditions:

$$\begin{aligned} r_l \leq r_s < r_u \\ |\overline{L_{1_A}S}| \leq R < |\overline{U_{1_A}S}| \end{aligned} \tag{A.23}$$

Area of coverage:

$$A = A_{AS\cap RS}(r_u, R, r_s) - A_{\pi_1}(R, |\overline{T_1T_2}|) - 2A_{\Sigma}(r_l, |\overline{L_{1_A}L_{1_B}}|) \tag{A.24}$$

Diagram:

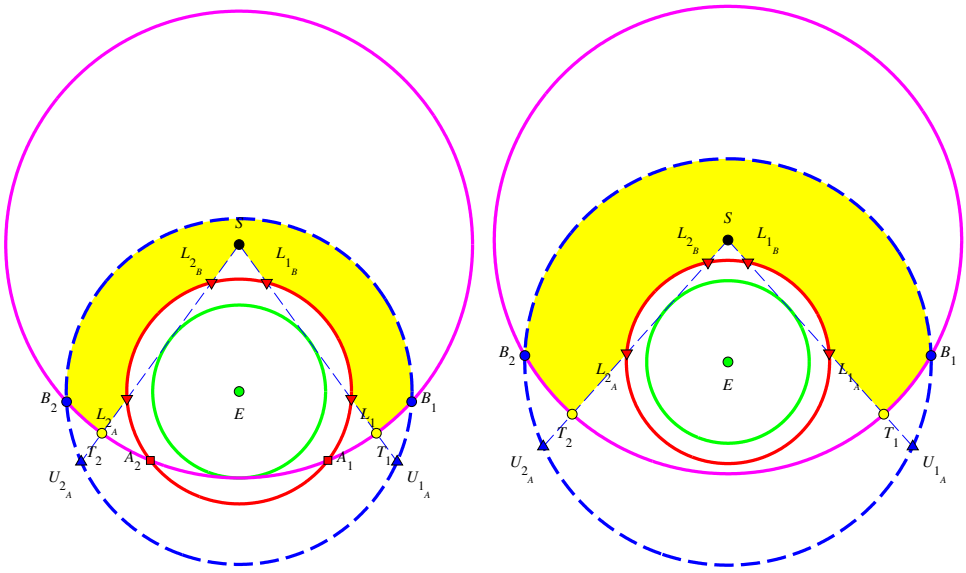


Figure A.13: Variations of Case 2c

A.4.2.4 Case 2d

Conditions:

$$\begin{aligned} r_l \leq r_s < r_u \\ |\overline{U_{1_A} S}| < R \end{aligned} \tag{A.25}$$

Area of coverage:

$$A = \pi r_u^2 - A_{\pi_2}(r_u, |\overline{U_{1_A} S}|, r_s) - 2A_{\Sigma}(r_l, |\overline{L_{1_A} L_{1_B}}|) \tag{A.26}$$

Diagram:

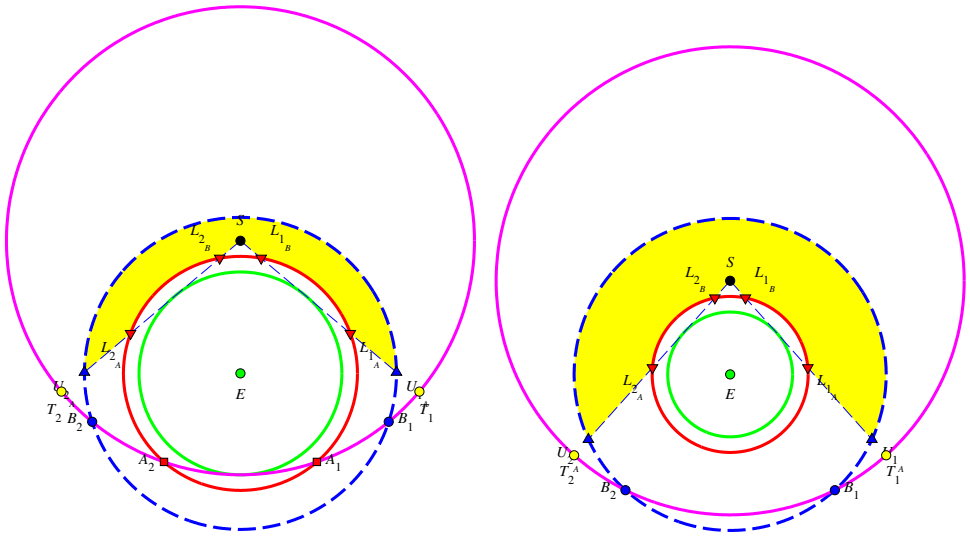


Figure A.14: Variations of Case 2d

A.4.3 Case 3 – $r_u \leq r_s$

A.4.3.1 Case 3a

Conditions:

$$\begin{aligned} r_u &< r_s \\ R &< |\overline{U_{1_B} S}| \end{aligned} \tag{A.27}$$

Area of coverage:

$$A = 0 \tag{A.28}$$

Diagram:

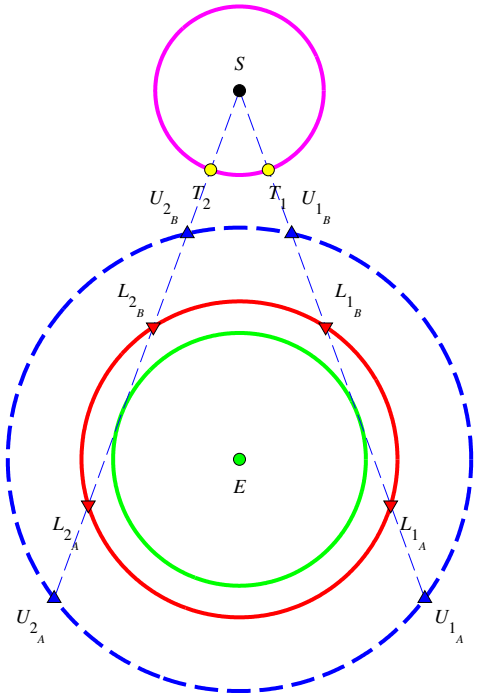


Figure A.15: Case 3a

A.4.3.2 Case 3b

Conditions:

$$\begin{gathered} r_u < r_s \\ |\overline{U_{1_B}S}| \leq R < |\overline{L_{1_B}S}| \end{gathered} \tag{A.29}$$

Area of coverage:

$$\begin{aligned} A = A_{AS\cap RS}(r_u, R, r_s) - A_{\pi_1}(R, |\overline{T_1T_2}|) \\ + A_{\Lambda_2}(r_u, |\overline{U_{2_B}S}|, |\overline{U_{2_B}U_{1_B}}|, |\overline{U_{1_B}S}|) \end{aligned} \tag{A.30}$$

Diagram:

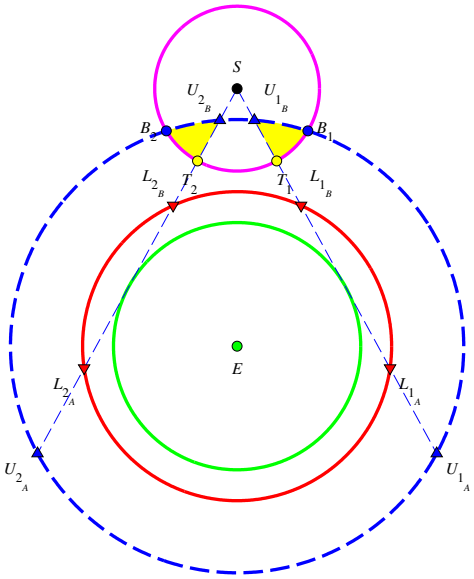


Figure A.16: Case 3b

A.4.3.3 Case 3c

Conditions:

$$\begin{aligned} r_u &< r_s \\ |\overline{L_{1_B}S}| \leq R &< |\overline{L_{1_A}S}| \end{aligned} \tag{A.31}$$

Area of coverage:

$$\begin{aligned} A = A_{AS\cap RS}(r_u, R, r_s) - A_{AS\cap RS}(r_l, R, r_s) \\ -A_{\Lambda_2}(r_l, |\overline{L_{2_B}S}|, |\overline{L_{1_B}L_{2_B}}|, |\overline{L_{1_B}S}|) \\ +A_{\Lambda_2}(r_u, |\overline{U_{2_B}S}|, |\overline{U_{1_B}U_{2_B}}|, |\overline{U_{1_B}S}|) \end{aligned} \tag{A.32}$$

Diagram:

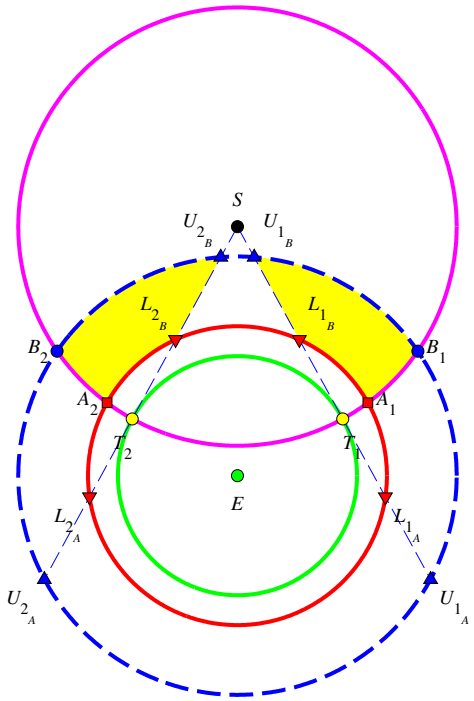


Figure A.17: Case 3c

A.4.3.4 Case 3d

Conditions:

$$\begin{gathered} r_u < r_s \\ |\overline{L_{1_A}S}| \leq R < |\overline{U_{1_A}S}| \end{gathered} \tag{A.33}$$

Area of coverage:

$$\begin{aligned} A = A_{AS\cap RS}(r_u, R, r_s) - A_{\pi_1}(R, |\overline{T_1T_2}|) \\ +A_{\Lambda_2}(r_u, |\overline{U_{2_B}S}|, |\overline{U_{1_B}U_{2_B}}|, |\overline{U_{1_B}S}|) \\ -2A_{\Sigma}(r_l, |\overline{L_{1_A}L_{1_B}}|) \end{aligned} \tag{A.34}$$

Diagram:

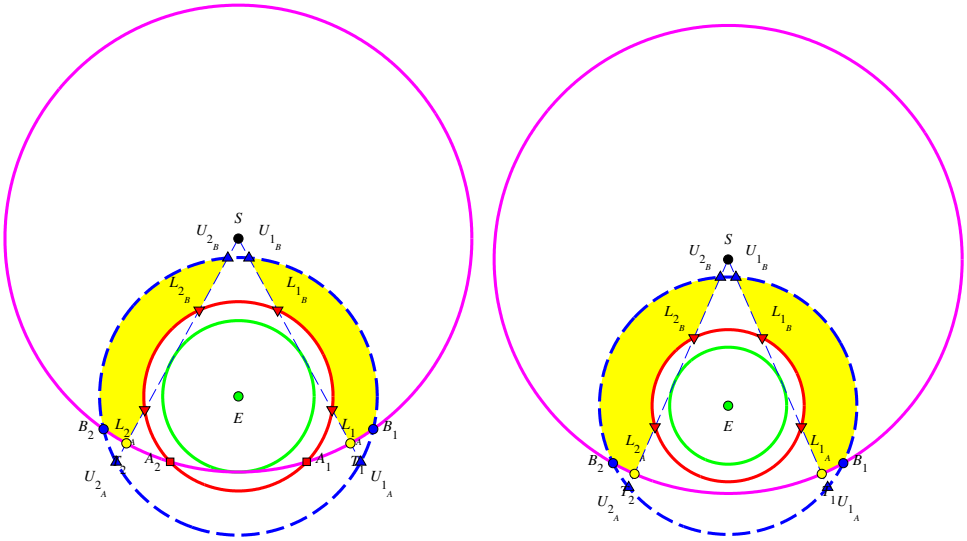


Figure A.18: Variations of Case 3d

A.4.3.5 Case 3e

Conditions:

$$\begin{aligned} r_u &< r_s \\ |\overline{U_{1_A}S}| &< R \end{aligned} \tag{A.35}$$

Area of coverage:

$$A = 2A_\Sigma(r_u, |\overline{U_{1_A}U_{1_B}}|) - 2A_\Sigma(r_l, |\overline{L_{1_A}L_{1_B}}|) \tag{A.36}$$

Diagram:

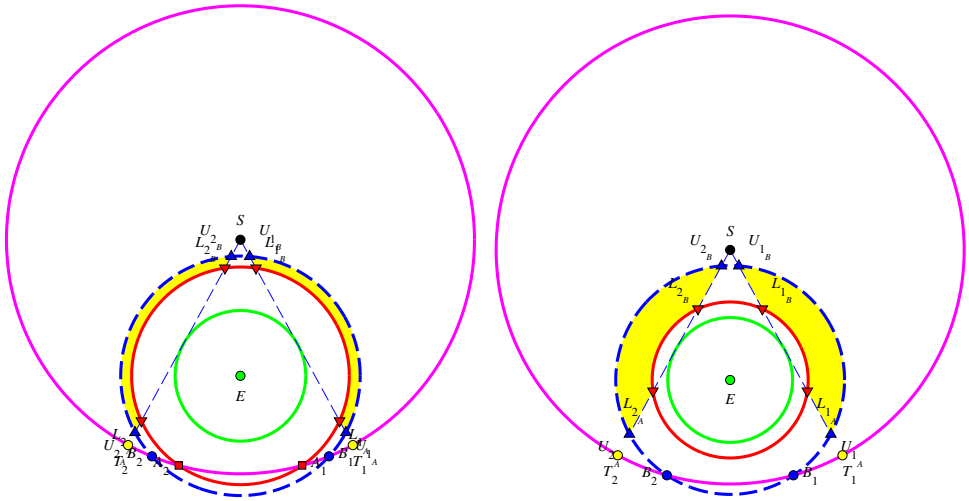


Figure A.19: Variations of Case 3e

A.5 Comparison with Published Expressions

Under direct comparison with the results published by Marchand and Kobel,[4] there are a number of minor differences between the coverage model sub-case expressions. In the formulation presented here, several of the published sub-cases are combined, yielding a smaller set of sub-cases. However, these differences are only the result of a different representation of the area of coverage, and do not change the resulting area calculation.

A.6 Implementation

A MATLAB function was developed, based directly on the results presented in this appendix, and extensive testing and reproduction of published results[4] demonstrated correctness. This script proves to be an invaluable validation tool for the numerical methods developed for this thesis, and is used as a basis for comparison in determining the amount of error introduced during numerical analysis of ATH coverage.

Appendix B

Additional Example Data

This appendix contains solutions and plots that may be of interest to the reader, but are omitted from previous chapters for brevity. Aside from some introductory remarks for each problem, these results are presented without additional discussion. However, their parameters and figures exactly fit within the context of the discussion from which they are drawn.

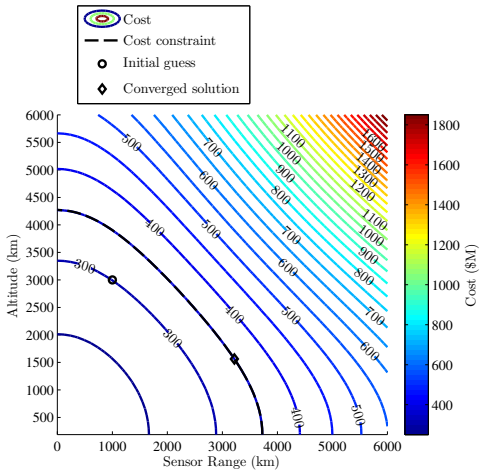
B.1 Chapter 4 – Example 1

The first example in Chapter 4 illustrates maximization of single coverage subject to a deployment budget constraint. The single and 10 satellite cases are discussed in Section 4.3, illustrating the impact upon constellation efficiency of redundant coverage (i.e. coverage at higher multiplicity than necessary). The 15 satellite constellation is of note because although it is allocated a deployment budget 40% larger than in the 10 satellite case, it delivers 1.5% less single coverage area. This hints at the result presented in Section 4.4 that, at least for the financial model developed in Section 4.1, smaller constellations with more robust sensors are more financially efficient.

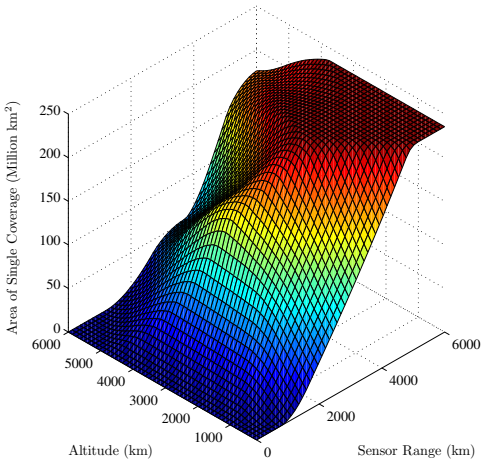
B.1.1 Example 1-B1 – 15 Satellites

Table B.1: Example 1-B1 – Parameters and Solution

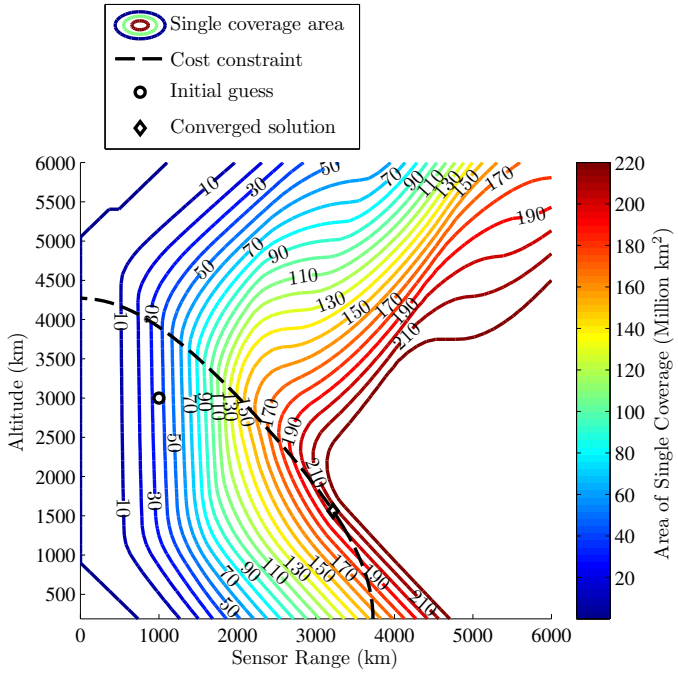
Parameter	Value	Description
n	15	number of satellites
$\Gamma_{\max}$	\$350M	budget constraint
R_0	1000 km	initial guess, R
h_0	3000 km	initial guess, h
R_{opt}	3218.487 km	converged area-optimal R
h_{opt}	1562.275 km	converged area-optimal h
$A_{1\times_{\text{opt}}}$	$205,406,521$ km^2	coverage area at solution
$\Gamma_{\text{opt}} - \Gamma_{\max}$	3.669×10^{-10} \$M	budget overrun



(a) Constraint Function (Financial Cost)

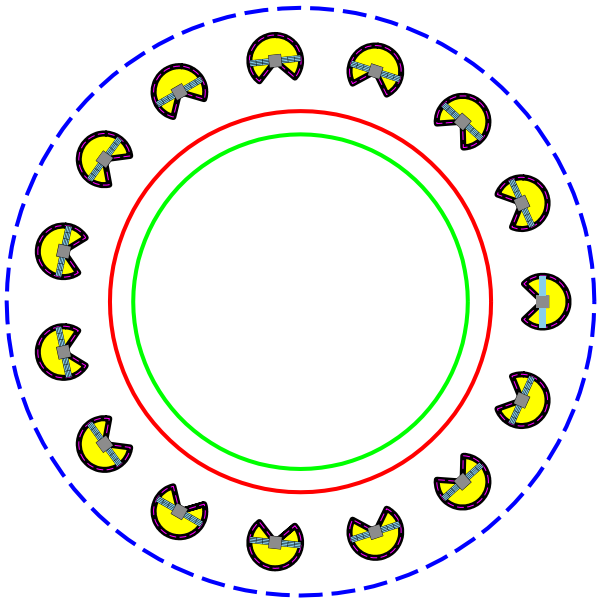


(b) Objective Function (Single Coverage Area)

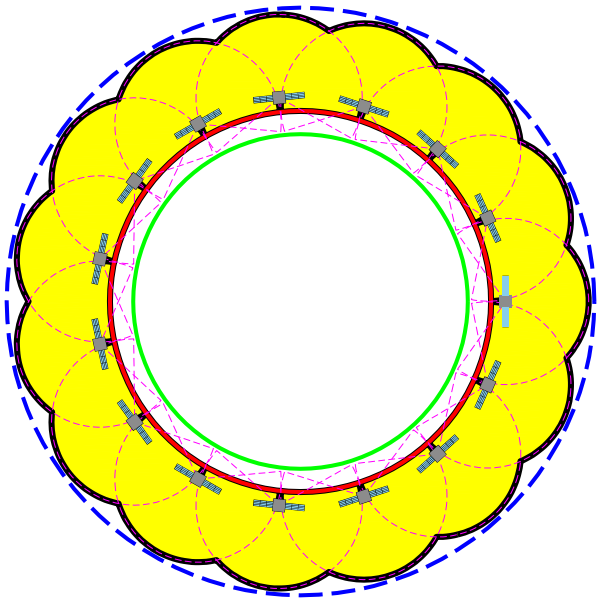


(c) Objective Function (Single Coverage Area) With Converged Solution

Figure B.1: Example 1-B1 – Constraint and Objective Contours



(a) Initial Guess



(b) Converged Solution

Figure B.2: Example 1-B1 – Initial Guess and Converged Solution

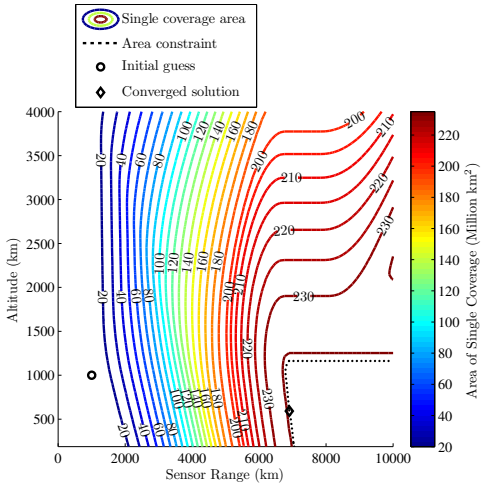
B.2 Chapter 4 – Example 2

Example 2 in Chapter 4 explores the minimization of constellation deployment budget subject to a constraint requiring 99.9% single coverage of the dual-altitude band area of interest. In Section 4.4.4, 3 through 10 satellite constellations are analyzed to determine their optimal costs in order to conclude that the 4 satellite configuration is the most cost-effective according to the financial model developed in Section 4.1. This section contains the 5 through 9 satellite cases, that are not explicitly discussed in Chapter 4.

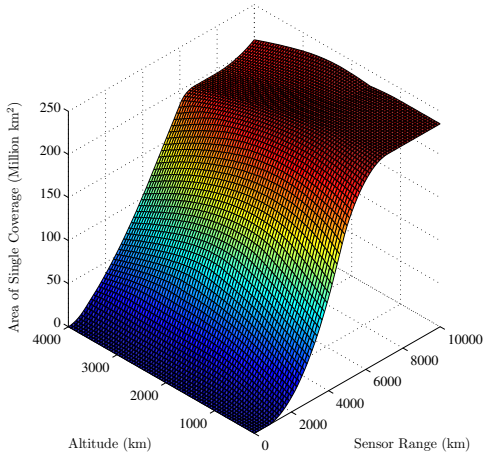
B.2.1 Example 2-B1 – 5 Satellites

Table B.2: Example 2-B1 – Parameters and Solution

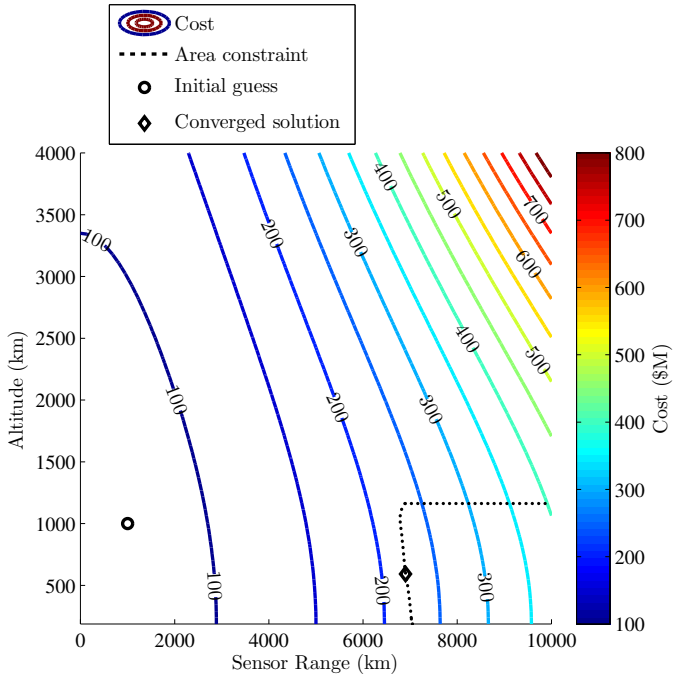
Parameter	Value	Description
n	5	number of satellites
$A_{\min}$	$0.999A_{AS}$	area constraint
R_0	1000 km	initial guess, R
h_0	1000 km	initial guess, h
R_{opt}	6900.591 km	converged cost-optimal R
h_{opt}	592.117 km	converged cost-optimal h
Γ_{opt}	220.595 \$M	converged cost at solution
$A_{1\times_{\text{opt}}} - A_{\min}$	9.222×10^{-4} km^2	coverage area surplus



(a) Constraint Function (Single Coverage Area)

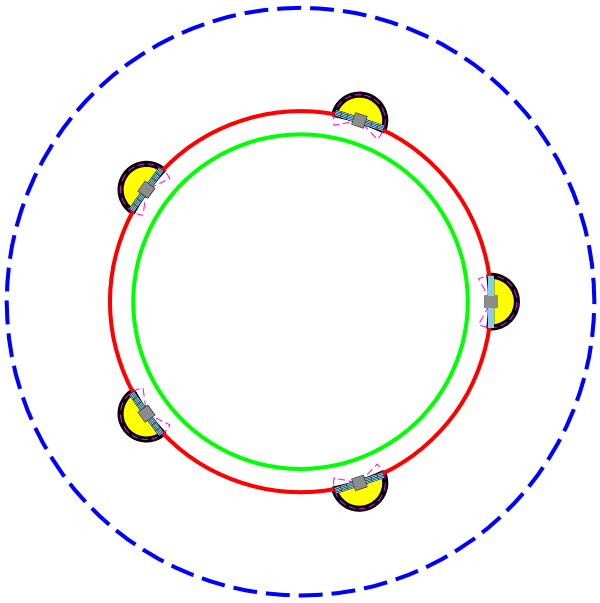


(b) Constraint Function (Single Coverage Area)

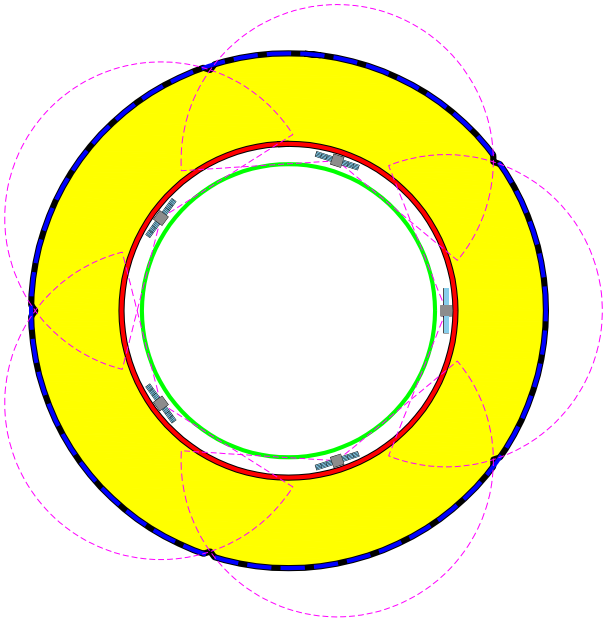


(c) Objective Function (Financial Cost)

Figure B.3: Example 2-B1 – Constraint and Objective Contours



(a) Initial Guess



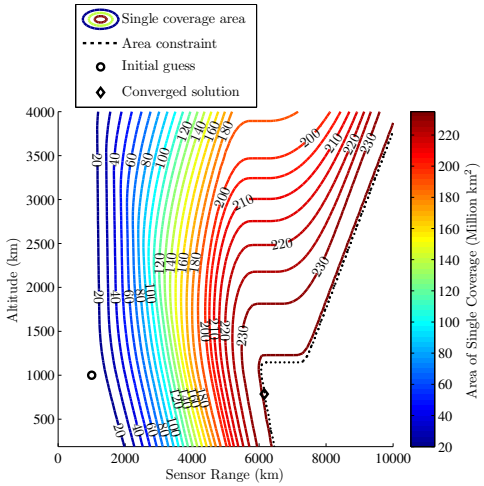
(b) Converged Solution

Figure B.4: Example 2-B1 – Initial Guess and Converged Solution

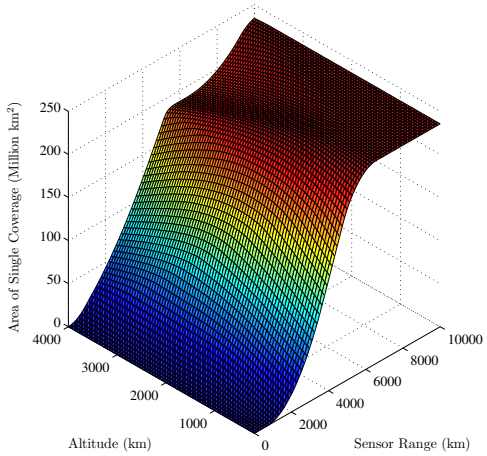
B.2.2 Example 2-B2 – 6 Satellites

Table B.3: Example 2-B2 – Parameters and Solution

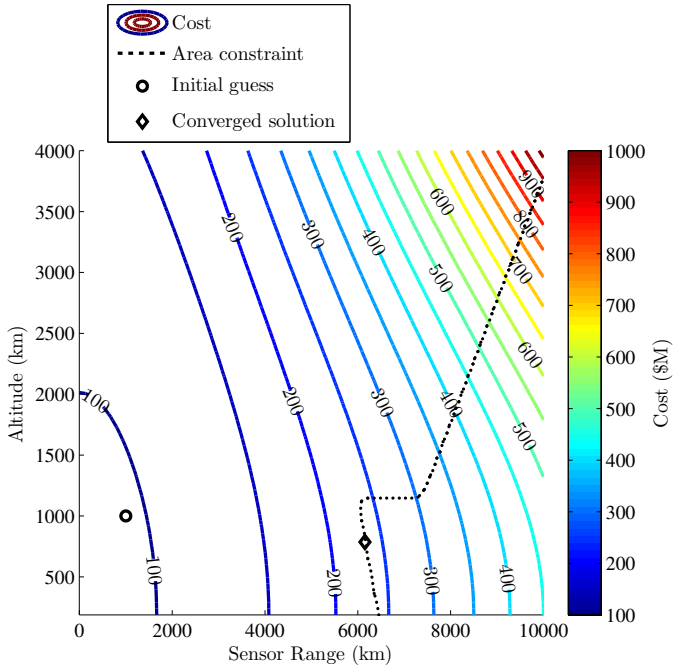
Parameter	Value	Description
n	6	number of satellites
$A_{\min}$	$0.999A_{AS}$	area constraint
R_0	1000 km	initial guess, R
h_0	1000 km	initial guess, h
R_{opt}	6154.896 km	converged cost-optimal R
h_{opt}	786.559 km	converged cost-optimal h
Γ_{opt}	232.339 \$M	converged cost at solution
$A_{1\times_{\text{opt}}} - A_{\min}$	-1.502×10^{-3} km^2	coverage area surplus



(a) Constraint Function (Single Coverage Area)

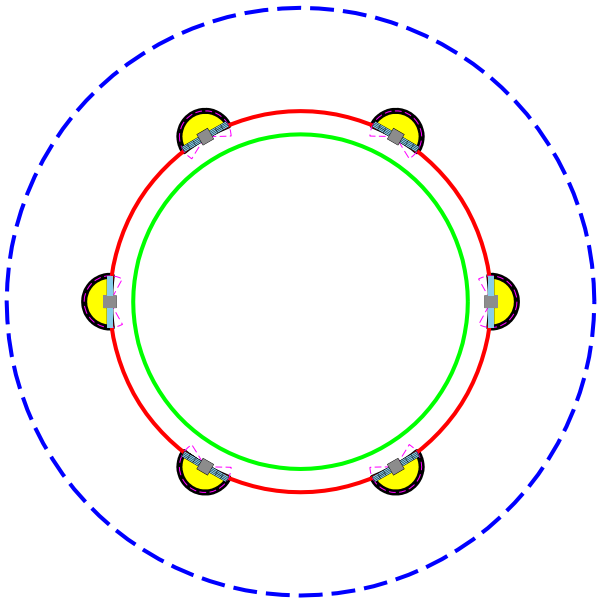


(b) Constraint Function (Single Coverage Area)

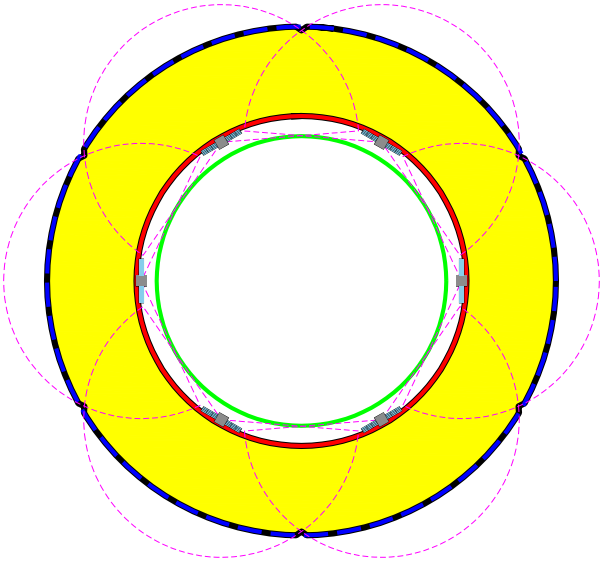


(c) Objective Function (Financial Cost)

Figure B.5: Example 2-B2 – Constraint and Objective Contours



(a) Initial Guess



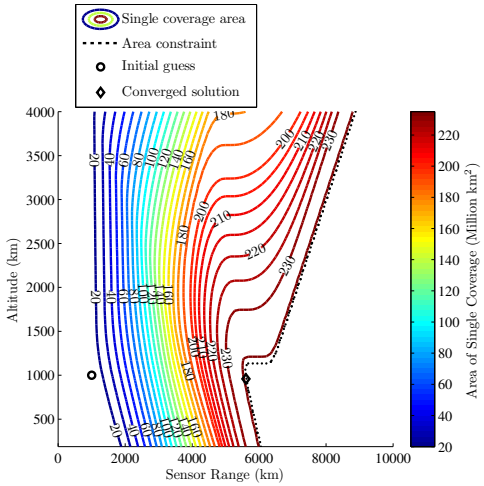
(b) Converged Solution

Figure B.6: Example 2-B2 – Initial Guess and Converged Solution

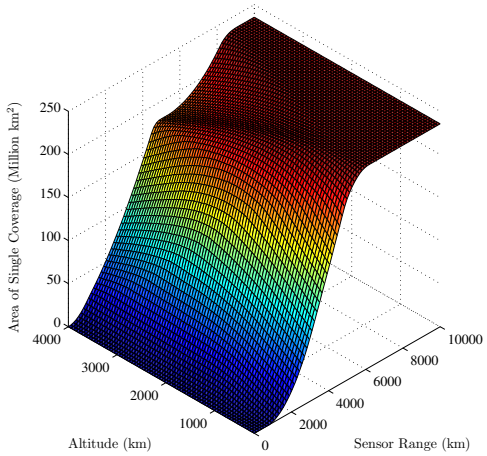
B.2.3 Example 2-B3 – 7 Satellites

Table B.4: Example 2-B3 – Parameters and Solution

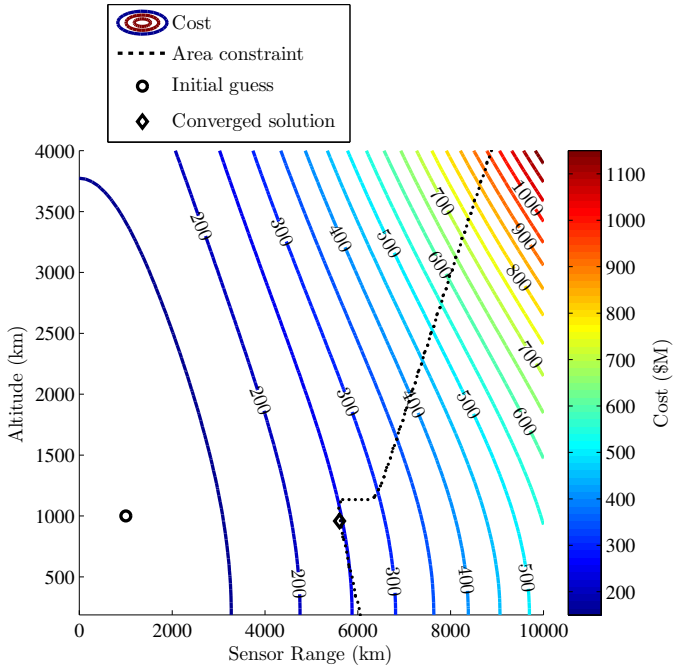
Parameter	Value	Description
n	7	number of satellites
$A_{\min}$	$0.999A_{AS}$	area constraint
R_0	1000 km	initial guess, R
h_0	1000 km	initial guess, h
R_{opt}	5606.081 km	converged cost-optimal R
h_{opt}	958.616 km	converged cost-optimal h
Γ_{opt}	246.915 \$M	converged cost at solution
$A_{1\times_{\text{opt}}} - A_{\min}$	-4.522×10^{-4} km^2	coverage area surplus



(a) Constraint Function (Single Coverage Area)

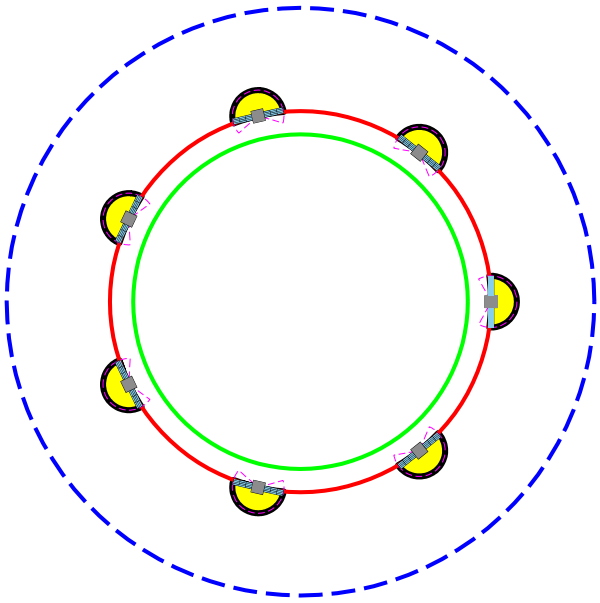


(b) Constraint Function (Single Coverage Area)

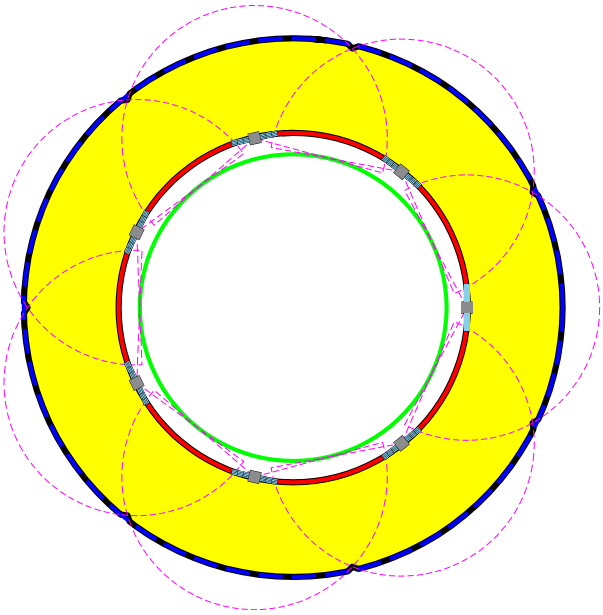


(c) Objective Function (Financial Cost)

Figure B.7: Example 2-B3 – Constraint and Objective Contours



(a) Initial Guess



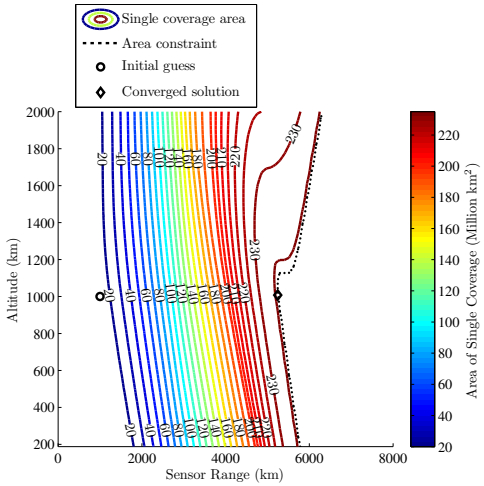
(b) Converged Solution

Figure B.8: Example 2-B3 – Initial Guess and Converged Solution

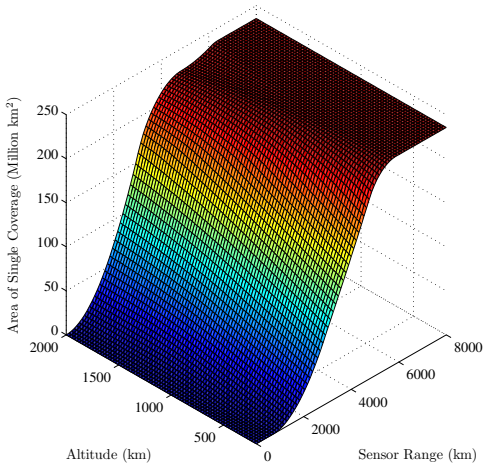
B.2.4 Example 2-B4 – 8 Satellites

Table B.5: Example 2-B4 – Parameters and Solution

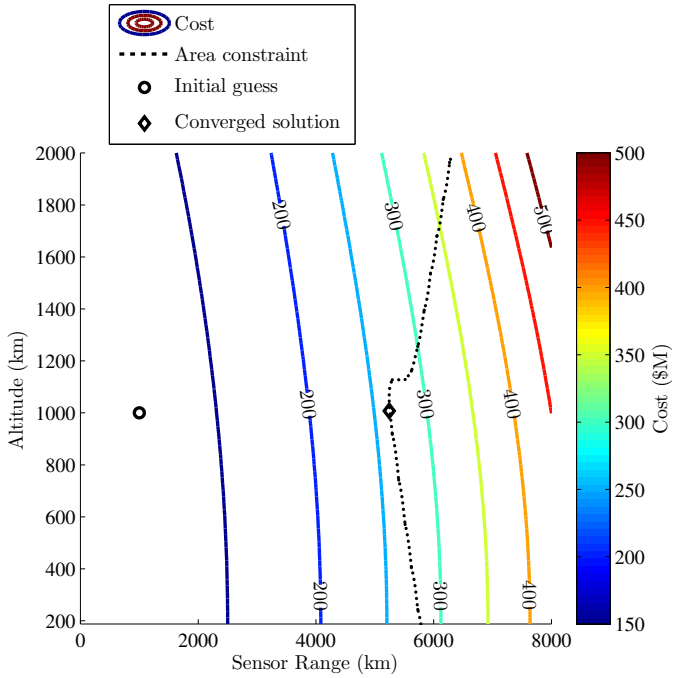
Parameter	Value	Description
n	8	number of satellites
$A_{\min}$	$0.999A_{AS}$	area constraint
R_0	1000 km	initial guess, R
h_0	1000 km	initial guess, h
R_{opt}	5243.835 km	converged cost-optimal R
h_{opt}	1007.459 km	converged cost-optimal h
Γ_{opt}	263.539 \$M	converged cost at solution
$A_{1\times_{\text{opt}}} - A_{\min}$	-6.613×10^{-5} km^2	coverage area surplus



(a) Constraint Function (Single Coverage Area)

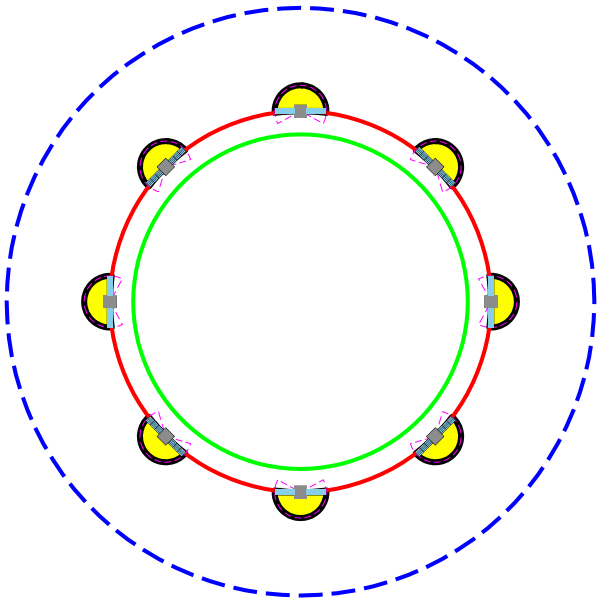


(b) Constraint Function (Single Coverage Area)

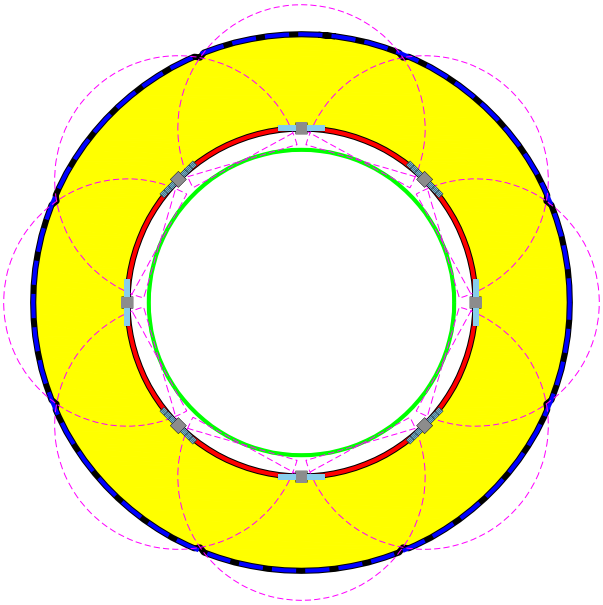


(c) Objective Function (Financial Cost)

Figure B.9: Example 2-B4 – Constraint and Objective Contours



(a) Initial Guess



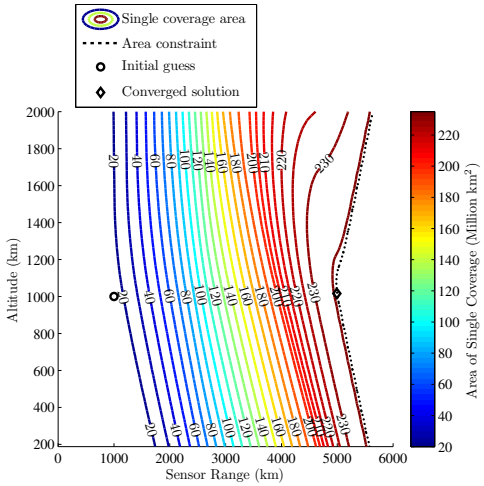
(b) Converged Solution

Figure B.10: Example 2-B4 – Initial Guess and Converged Solution

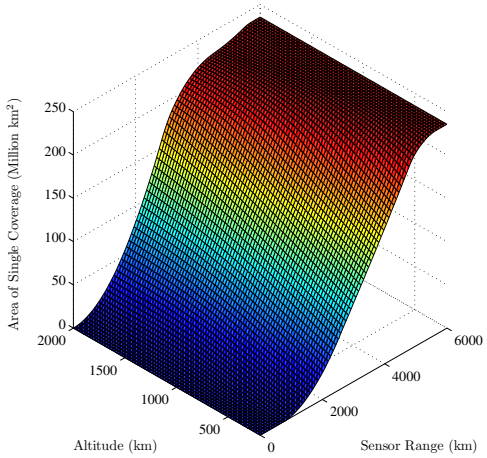
B.2.5 Example 2-B5 – 9 Satellites

Table B.6: Example 2-B5 – Parameters and Solution

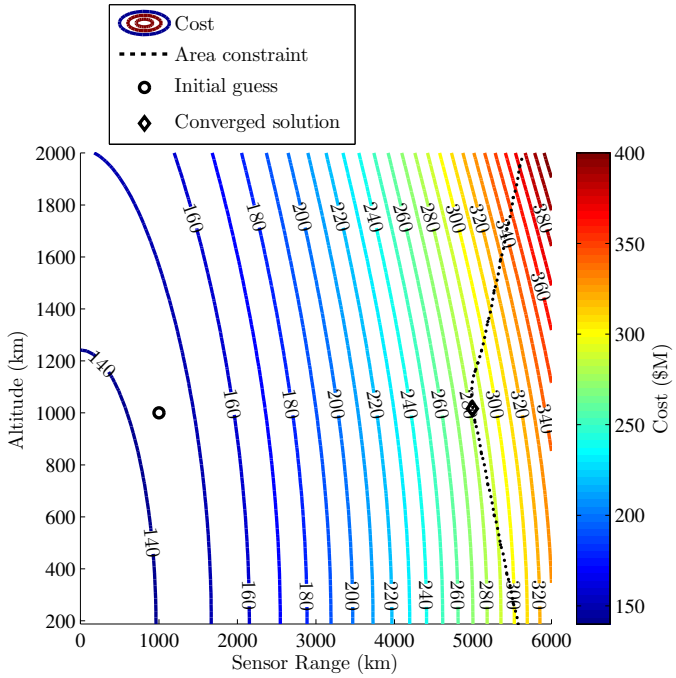
Parameter	Value	Description
n	9	number of satellites
$A_{\min}$	$0.999A_{AS}$	area constraint
R_0	1000 km	initial guess, R
h_0	1000 km	initial guess, h
R_{opt}	4993.666 km	converged cost-optimal R
h_{opt}	1016.415 km	converged cost-optimal h
Γ_{opt}	281.988 \$M	converged cost at solution
$A_{1\times_{\text{opt}}} - A_{\min}$	-3.296×10^{-5} km^2	coverage area surplus



(a) Constraint Function (Single Coverage Area)

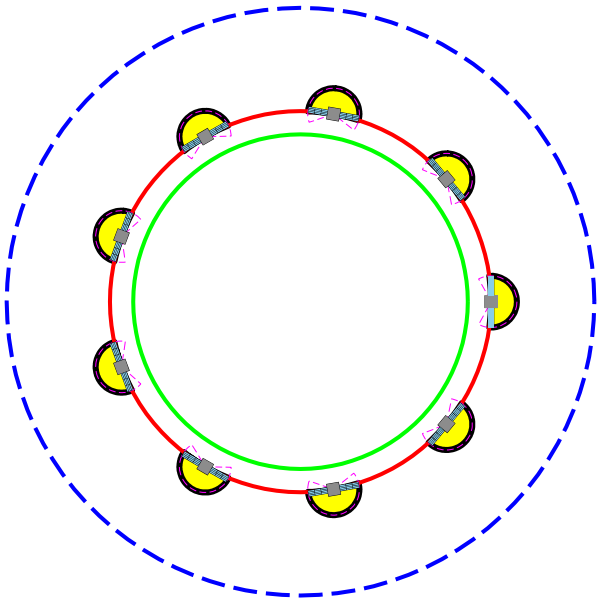


(b) Constraint Function (Single Coverage Area)

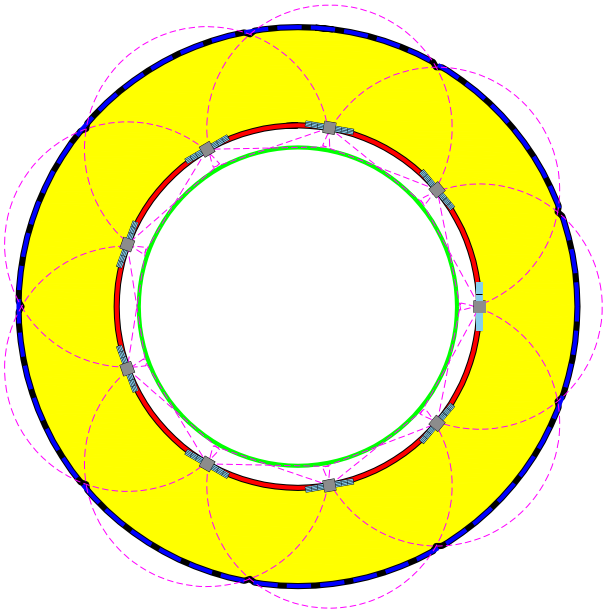


(c) Objective Function (Financial Cost)

Figure B.11: Example 2-B5 – Constraint and Objective Contours



(a) Initial Guess



(b) Converged Solution

Figure B.12: Example 2-B5 – Initial Guess and Converged Solution

Bibliography

[1] L. Rider, "Design of low to medium altitude surveillance systems providing continuous multiple above-the-horizon viewing," *Optical Engineering*, vol. 28, no. 1, pp. 25–29, 1989.

[2] F. Martinez, A. J. Rueda, and F. R. Feito, "A new algorithm for computing boolean operations on polygons," *Computers & Geosciences*, vol. 35, pp. 1177–1185, 2009.

[3] G. Jacquenot, "Polygon intersection," http://www.mathworks.com/matlabcentral/fileexchange/18173-polygonintersection.

[4] B. G. Marchand and C. Kobel, "Geometry of optimal coverage for space based targets with visibility constraints," *The Journal of Spacecraft and Rockets*, vol. 46, no. 4, pp. 845–857, 2009.

[5] *Polybool*, http://www.mathworks.com/help/toolbox/map/ref/polybool.html, The MathWorks.

[6] A. Murta, "General polygon clipping library," http://www.cs.man.ac.uk/~toby/alan/software/.

[7] S. Holz, "Polygon clipper," http://www.mathworks.com/matlabcentral/fileexchange/8818-polygonclipper.

[8] J. R. Wertz and W. J. Larson, Eds., *Space Mission Analysis and Design*, 3rd ed. Microcosm Press, 2007, ch. 20.

[9] L. Rider, "Optimal orbital constellations for global viewing of targets against a space background," *Optical Engineering*, vol. 19, no. 2, pp. 219–223, 1980.

[10] D. C. Beste, "Design of satellite constellations for optimal continuous coverage," *IEEE Transactions on Aerospace and Electronic Systems*, vol. 14, no. 3, pp. 466–473, 1978.

[11] J. M. Hanson and A. N. Linden, "Improved low-altitude constellation design methods," *Journal of Guidance, Control, and Dynamics*, vol. 12, no. 2, pp. 228–236, 1988.

[12] K. J. Gordon, "The computation of satellite constellation range characteristics," in *AIAA/AAS Astrodynamics Conference*, August 1994.

[13] L. Rider and W. S. Adams, "Circular polar constellations providing continuous single or multiple coverage above a specified latitude," *Journal of the Astronautical Sciences*, vol. 35, pp. 155–192, 1987.

[14] J. G. Walker, "Circular orbit patterns providing continuous whole earth coverage," Royal Aircraft Establishment, Tech. Rep. 70211, November 1970.

[15] ——, "Continuous whole earth coverage by circular orbit satellite patterns," Royal Aircraft Establishment, Tech. Rep. 77044, March 1977.

[16] M. de Berg, O. Cheong, M. van Kreveld, and M. Overmars, *Computational Geometry: Algorithms and Applications*, 3rd ed. Springer-Verlag, 2008.

[17] B. Vatti, "A generic solution to polygon clipping," *Communications of the ACM*, vol. 35, no. 7, pp. 57–63, 1992.

[18] G. Greiner and K. Hormann, "Efficient clipping of arbitrary polygons," *ACM Transactions on Graphics*, vol. 17, no. 2, pp. 71–83, 1998.

[19] W. H. Beyer, Ed., *CRC Standard Mathematical Tables*, 28th ed. CRC Press, 1987, pp. 318–321.

[20] K. Weiler, "Polygon comparison using a graph representation," in *SIGGRAPH 80*, 1980, pp. 10–18.

[21] A. Johnson, "Clipper," http://sourceforge.net/projects/polyclipping/.

[22] D. Kennison, "Polypack, a package of routines to manipulate polygons," http://www.dkrz.de/ngdoc/ng/supplements/polypack/.

[23] J. L. Bentley and T. A. Ottmann, "Algorithms for reporting and counting geometric intersections," *IEEE Transactions on Computers*, vol. C-28, no. 9, pp. 643–647, 1979.

[24] Y. K. Liu, X. Q. Wang, S. Z. Bao, M. Gombosi, and B. Zalik, "An algorithm for polygon clipping, and for determining polygon intersections and unions," *Computers & Geosciences*, vol. 33, pp. 589–598, 2007.

[25] W. H. Beyer, Ed., *CRC Standard Mathematical Tables*, 28th ed. CRC Press, 1987, pp. 123–124.

[26] *Polyarea*, http://www.mathworks.com/help/techdoc/ref/polyarea.html, The MathWorks.

[27] S. B. Lippman, J. Lajoie, and B. E. Moo, *C++ Primer*, 4th ed. Addison Wesley, 2005, ch. 5, p. 177.

[28] R. R. Bate, D. D. Mueller, and J. E. White, *Fundamentals of Astrodynamics*. Dover Publications Inc., 1971.

[29] *Fmincon*, http://www.mathworks.com/help/toolbox/optim/ug/fmincon.html, The MathWorks.

[30] M. Powell, "A fast algorithm for nonlinearly constrained optimization calculations," in *Numerical Analysis*, ser. Lecture Notes in Mathematics, G. Watson, Ed. Springer Berlin / Heidelberg, 1978, vol. 630, pp. 144–157.

[31] *ImageDIG*, http://www.imagedig.com, SciCepts Engineering.

[32] H. W. Kuhn and A. W. Tucker, "Nonlinear programming," in *Proceedings of the 2nd Berkeley Symposium*. University of California Press, 1951, pp. 481–492.

[33] D. H. Martin, "The essence of invexity," *Journal of Optimization Theory and Applications*, vol. 47, pp. 65–76, 1985.

[34] W. Dunham, *Journey through Genius: The Great Theorems of Mathematics*. Wiley, 1990, ch. 5, pp. 113–132.